NPS-CS-07-006

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

Proceedings of the
First IEEE International Workshop on Safety of Systems

By

James Bret Michael
John Hauraz
Zachary Pace

Prepared for: Center for National Software Studies
NAVAL POSTGRADUATE SCHOOL
Monterey, California 93943-5000

NAVAL POSTGRADUATE SCHOOL
Monterey, California 93943-5000

Daniel T. Oliver
President

Leonard A. Ferrari
Provost

This report was prepared for the First IEEE International Workshop on Safety of Systems.
Reproduction of all or part of this report is authorized.

This report was prepared by:

_____
James Bret Michael
Professor of Computer Science and Electrical Engineering
Naval Postgraduate School

Reviewed by:

_____
Peter J. Denning, Chairman
Department of Computer Science

Released by:

_____
Dan C. Boger
Interim Associate Provost and
Dean of Research

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** 7/19/07	**3. REPORT TYPE AND DATES COVERED** Technical Report	
4. TITLE AND SUBTITLE: Proceedings of the First IEEE International Workshop on Safety of Systems			**5. FUNDING NUMBERS**
6. AUTHOR(S) James B. Michael, John Hauraz, Zachary Pace			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER** NPS-CS-07-006
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, California 93943			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this technical report are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited.			**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(maximum 200 words)***

In March 2007, the Technical Committee on System Safety (TCSS) under the IEEE System Council held its first international workshop on issues relating to safety of systems of national and global significance. The workshop provided an open working forum to contribute to the goal of obtaining a holistic view of system safety for a better understanding of the system safety discipline. Much of the discussion that took place during the workshop revolved around what are the relationships between safety and the other areas of dependability, such as security and reliability, and how to leverage these relationships to build trustworthy systems. This report contains the position papers along with presentation material delivered by the participants of the workshop, along with a summary of the participants' input from the discussion portion of the workshop.

14. SUBJECT TERMS Software, Security, Safety			**15. NUMBER OF PAGES** 174
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

Standard Form 298 (Rev.2-89)

THIS PAGE IS INTENTIONALLY LEFT BLANK

Proceedings for the First IEEE International Workshop on Safety of Systems.
Sponsored by the Technical Committee on System Safety of the IEEE Systems Council

Naval Postgraduate School, Monterey, California, Mar. 15–16, 2007

THIS PAGE IS INTENTIONALLY LEFT BLANK

Preface

In March 2007, the Technical Committee on System Safety (TCSS) under the IEEE System Society held its first international workshop on issues relating to safety of systems of national and global significance. The workshop provided an open working forum to contribute to the goal of obtaining a holistic view of system safety for a better understanding of the system safety discipline. Because technical societies contain expertise in various slices of the discipline of system safety, obtaining both the big picture and coordination among the separate initiatives is difficult to achieve in any one specific society. We envision the workshop becoming a unique integrating forum for researchers and practitioners to discuss the practical issues associated with safety of systems.

The call for participation included the following list of topics:

- Safety engineering of systems-of-systems;
- Building a safety culture and management of safety;
- Education and teaching materials;
- Safety Standards;
- Ethics;
- Competency of safety practitioners;
- Human factors and ergonomics including psychological aspects of safety;
- Assessment of safety and development of safety cases;
- Development of the safety requirements to identify the safety functions to be performed and identification of the safety integrity of the various safety functions;
- Hazard analysis and risk assessment techniques as a basis of the development of the safety requirements specification;
- Design and implementation considerations;
- Modeling and formal methods of assurance including accident modeling;
- Effective and appropriate use of tools

The attendees provided input on the following questions:

What are some of the fundamental knowledge barriers for safety of systems today?

- How to measure safety, security, and other ilities for stovepipe and system of systems
- How to make weightings explicit for tradeoff analysis, and are those the correct weights
- Need for both concepts and definitions to be understood by safety, security, and other communities
- We are unable to describe uncertainty in common terms
- Misunderstanding of what standards provide
- Practitioner competence
- Realistic expectations on practitioners
- Risk management, such as how to model security problems

- Understanding the roles and responsibilities of each discipline, and how they fit together
- What decisions are we trying to support from our analyses of systems

What are some of the fundamental limitations for safety of systems today?

- A mindset of evolving vice building dependable systems
- Influences of organizational culture and established work practices
- Problem-solving approaches resulting in unnecessarily complex systems
- Lack of integration among policy, guidance (how to do it), standards and compliance enforcement
- Defining the system boundary
- Lack of codification within standards
- Unknowns: very large number of possible vulnerabilities, hazards, etc.
- Incentives are not congruent with the risks; identify what causes those factors to be in the decision formula (not defined in the standards today)
- System integration is done poorly, partly due to the lack of tool support
- Turf issues, such as between IEEE technical committees, societies, and councils

What are the most important research challenges?

- There needs to be an as-is report of the safety and security domains
- Create the to-be report for both the safety and security domains, including the mission and sustainment domain
- Perform the gap analysis
- Assurance cases
- Automation support for building and analysis of architectures on an ility-basis
- Composition of systems into system of systems, including across organizations
- How do you specify uncertainty for security?
- Establish a sub grand challenge on dependability

What are promising innovations and abstractions for building future high-confidence safety systems?

- Assurance cases that are usable across domains
- Tools interoperability: tools that reuse existing data rather than rely on translating data between tools, analyses, etc.
- Formalize system of systems engineering techniques, concepts, etc.
- Formalize the as-is availability and data trades between safety and security
- Formally codify precepts (programmatic, design, operations guidelines) for both safety and security, and cross compare
- Encourage IEEE headquarters to foster cooperation across societies, councils, technical committees to address system dependability
- Encourage the IEEE Computer Society, International System Safety Society, and RAMS to re-establish joint conferences between safety and security; for international coverage: SAFECOM, IEE Software Safety Symposium

What are possible milestones for the next 5 years?

- Finalize the
  - As-is report of the safety and security domains
  - To-be report for both the safety and security domains, including the mission and sustainment domain
  - Gap analysis
- Standards on assurance that span safety, security and other aspects of dependability, such as ISO/IEC 15026, and safety standards such as AOP 52 and MIL-STD-882
- Have a outline (or roadmap) of the body of knowledge—provide help to the engineering, program manager and others on how to and what to apply develop dependable systems
- Making the accreditation more standard and visible
- Have a body of knowledge for assurance, in addition have a breakdown of skill sets against roles
- Have cooperation with the IEEE Product Safety Engineering Society and other societies to build a safety-security accreditation program

What are possible milestones for the next 5-to-10 years?

- Risk-decisions are made across all types of risk, risks throughout the lifecycle
- User high-quality software engineering methodologies
- Meet much higher expectations for dependability of systems (i.e., ultra-high dependability)—raise the bar

What in the near term can IEEE do for the attendees of the workshop and members of the technical meeting?

- Establish avenues for members of the community of interest (COI) on dependability to share ideas and documents
- Establish a column editor for Security & Privacy, Software, or some other IEEE magazine to address the role between security and safety, with articles being on the order of 2000 words

In addition, the attendees of the workshop provided input to the TCSS roadmap. Suggested themes for the next workshop included the following:

- Applying autonomic and biological computing (e.g., swarms) to address safety and security
- Certification of systems and people
- Roadmap—address what was brought up during the first workshop
- Relate safety and security to one another in the system-dependability context
- Look into processes
- Facilitation of communication between security and safety practitioners

The attendees also recommended that the technical committee do the following:

- Identify interests of TC members
- Start a reading list

- Encourage weekly posting of definitions and concepts for feedback—set up a wiki for the TC
- Invite papers that address integration of safety and security
- Put together a panel: Can safety and security be hooked up: Is there any relationship between the two

The next workshop will be held in conjunction with the Second Annual IEEE Systems Conference, April 7-10, 2008 in Montreal, Quebec, Canada.

J. Bret Michael
Chair, IEEE Technical Committee on System Safety
General Chair, IWSS

John Hauraz
Founder, IEEE Technical Committee on System Safety
Finance Chair, IWSS

Summary of Position Papers

Selected Issues in Computer Systems Safety: Position Paper, *Andrew J. Kornecki, Janusz Zalewski*

This paper addresses the role of software in system safety, where the application of computers or programmable devices may put the users or public at risk.

To make significant progress inventing and innovating in the area of safety assessment and assurance there will need to be a corresponding level of funding to similar to steps taken a few years ago to sponsor security research.

The way the present authors see progress made possible in the next 5-10 years is via a significant coordinated effort of respective government agencies and industrial sectors. It should be made clear to the decision makers that if cost minimization will continue to be an essential factor in safety-related industries, then we may soon experience the kind of failures which were caused not so long ago by breaches in security.

Subject Introduction, *Archibald McKinlay*

This introduction is background for three papers which require a similar introduction:

- Hooking into Systems Engineering
- Systems Safety Engineering HR
- Systems Safety in new Architecture and Technologies

Unlike Systems Safety Engineering, little has been done to incorporate software requirements and risk management into Systems Engineering. There needs to be a holistic systems integration approach to the updating of Systems Engineering to re-integrate Systems Safety Engineering, Systems Assurance and Security, and Software Engineering. The DoD has efforts in-work to update the Systems Engineering Plan (SEP) but it will be a year more before it is finished.

Each added discipline required a change in the typical engineer's abilities, education and experience. The advancing technologies must be viewed in the same model. When a safe system is taken and simply attached to the Internet for monitoring the safety risk changes, and changes in ways that are not obvious to the traditional Software Safety or Systems Safety Engineer. The technologies are changing so fast that systems are being built right now without the updated training, education, or toolkit being available because neither the chip nor the interface existed at the project's start.

Systems Engineering must return to the roots of risk management and use that to maintain focus in prioritizing tasks in all schedules, meetings and budgets. Like Systems Safety was made to absorb occupational and then environmental tasks, so also must Systems Engineering reconnect to its many children. All children must coordinate through and with Systems Engineering.

Safety and Security in Software Engineering, *Samuel T. Redwine, Jr.*

Today, security is a concern for most systems as software has become central to the functioning of organizations and physical systems with much of it directly or indirectly exposed to the Internet or to insider attack as well as to subversion during development, deployment, and updating. Though safety-oriented systems so exposed now must also face the security problem, often traditional computing safety engineering does not address maliciousness and its perverse effect on probabilistic analyses.

One can write requirements for safety and security that look quite similar in form. When both safety and security are required, a number of areas are candidates for combining safety and security engineering concerns including: goals or claims including sub-goals or subclaims, constraints on system behavior, principles, solutions, activities and processes, assurance cases, correctness and evaluations and certifications.

In recent years, the software safety community has been more advanced in its thinking and has more examples of successful experience with producing high-confidence software than does the software security community. The safety community's experience provides lessons for software security practitioners, but the traditional engineering safety problem differs from the security one in a critical way—it presumes non-existence of maliciousness.

Competency Software Safety Requirements for Navy Engineers, *Brian Scannell, Paul Dailey*

The Navy currently has no formal certification for Safety Engineers concentrating in software safety. NOSSA has led an effort to educate personnel regarding the development and support of Naval Weapon Systems. The WISE training tool is a step in the right direction, but further formal training is needed to support experience in software safety. There is a need to evade the case of untrained software safety engineers that are arbitrarily appointed tasks. This should not be based on education or experience alone but rather a combination of experience, education, and certification. A documented certification process will only improve systems required to be safe that depend on software. There are several options to obtain a solid software and systems safety background for Navy applications.

Formal training is important, and provides a good foundation for software safety. In addition, certified software safety engineers should also have a mentor assigned to them, attend safety board presentations to gain experience, and also provide a report of training experiences.

Biologically-Inspired Concepts for Autonomic Self-Protection in Multiagent Systems, *Roy Sterritt, Mike Hinchey*

Biologically-inspired autonomous and autonomic systems (AAS) are essentially concerned with creating self-directed and self-managing systems based on metaphors from nature and the human body, such as the autonomic nervous system. Agent technologies have been identified as a key enabler for engineering autonomy and autonomicity in systems, both in terms of retrofitting into legacy systems and in designing new systems. Handing over responsibility to systems themselves raises concerns for humans with regard to safety and security.

Autonomic agents have been gaining ground as a significant approach to facilitate the creation of self-managing systems to deal with the ever increasing complexity and costs inherent in today's and tomorrow's systems.

In terms of the Autonomic Systems initiative, agent technologies have the potential to become an intrinsic approach within the initiative , not only as an enabler, but also in terms of creating autonomic agent environments.

Toward a Unified Safety/Security Model, *Gary Stoneburner*

The worlds of safety and security have co-existed for some time, yet remain largely separate domains with limited interactions. This is, to put it mildly, a problem. Each domain has contributions for the other and more dependable systems being a significant benefit of working together. Yet one element that has continued to separate these domains is lack of a common language and taxonomy for discussing risks associated with safety and with security.

Both safety and security have much to gain by working together. Security can piggy-back on the work done within the safety community in developing definitions and terminology to express hazard conditions and in establishing organizational awareness of the need to trade function for other, important concerns. The safety community can take advantage of the work done by security in dealing with intelligent maliciousness which is not well addressed by the probabilistic assumptions that underlie safety processes and yet is now a significant concern with regard to safety. A risk framework has been proposed to help make the idea of working together more than just an idea, but a reality.

THIS PAGE IS INTENTIONALLY LEFT BLANK

Table of Contents

THIS PAGE IS INTENTIONALLY LEFT BLANK

Selected Issues in Computer Systems Safety: Position Paper

Andrew J. Kornecki
Dept. of Computer & Software Engineering
Embry-Riddle Aeronautical University
Daytona Beach, FL 32114-3900, USA
kornecka@erau.edu

Janusz Zalewski
Computer Science Department
Florida Gulf Coast University
Fort Myers, FL 33965-6565, USA
zalewski@fgcu.edu

Abstract

The position paper presents the authors' views on the critical issues in safety of computer systems and software. It is based on selected results from several studies the authors have done for various government agencies, private companies and professional societies. Main limitations and challenges in designing computer systems for safety are discussed.

1. Introduction

System safety is a very broad term and books have been written on various aspects of safety analysis and safety assurance [1,2]. In this position paper, we are focusing in particular on various aspects of computer safety, especially the role of software in system safety, where the application of computers or programmable devices may put the users or public at risk. The authors' experience comes mostly from research related to aviation, air transportation and space, but partially also from research on medical, automotive and nuclear devices and technologies. However they by no means claim that the treatment of the subject is complete and exhaustive.

In a broader sense, to evaluate safety of a computer product, especially the software product that is used in a safety critical system, one has to take a closer look at a product itself, but also at the way it has been developed, as well as at the way the tools for developing this product have been created. This logic is illustrated in Figure 1, and is very different from the traditional approaches to system safety, where the analysis is limited only to the product and the related application environment.

The examples come from the recent study on the assessment of software development tools for safety-critical real-time systems conducted for the Federal Aviation Administration (FAA) [3]. Modern commercial development tools are typically complex suites combining multiple functionalities. Considering tool complexity, the quality of support materials is often marginal. Unless developers become expertly proficient with the tool, reliance on it may lead to ignorance of tool functionality, complacency and thus compromise the safety of developed system.

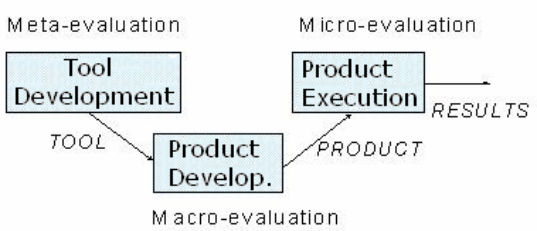


Fig. 1. Context for Evaluating Computer Products.

2. Limitations and Knowledge Barriers

What are the three fundamental limitations and knowledge barriers for safety of systems today?

From the computer use and software standpoint, there are several issues that obstruct progress in dealing with safety. The most important among them seem to be the following:

1) Limited understanding of computers and software by safety engineers and, vice versa, limited understanding of safety issues by computer and software engineers.
2) Very confusing state of safety standards and guidelines, and proliferation of sometimes contradicting guidelines. This situation results in the sheer number of documents the safety critical system developers must be aware of.
3) Lack of well-defined, measurable safety metrics is another fundamental limitation to progress in safety assurance.

Our studies based on the safety related software guidelines in civil aviation DO-178B [4] indicated that the criteria used in this and other safety related standards do not include solid theoretical underpinnings to be used as measures of metrics for safety. This is a significant impediment in product qualification and certification [5].

3. Research Challenges

What are the three most important research challenges?

As it stands right now, even agreeing on the state of the art and practice in computer and software safety research would be difficult. One important step forward would be to produce a document defining the *body of knowledge* in computer system safety, similar to the one produced for security [6]. This would help establish the common ground, from which further steps could be possibly defined. The challenges that researchers are facing in this respect, come from at least the following:

1) Lack of specific data typically available from industrial projects, since the industry does not share this type of data due to the competitive advantage.
2) Common-off-the-shelf components (COTS), both hardware and software, are going to be increasingly used in safety critical systems, but very few studies have been done how to approach their safety assessment.
3) New technologies will proliferate, both in hardware, such as high speed databuses [7], and software, such as automatic code generation [8], for the analysis of which new research methods and approaches will have to be created.

From the perspective of our studies, a critical issue for vendors and government agencies was the necessity of certification based on solid experimental data. However, the qualification data collected from experiments constitute a component of the certification package and are highly proprietary. This situation puts researchers in a very disadvantageous position. Some relevant discussions how to address this and similar issues, have recently taken place at the Tools Forum [8].

4. Promising Innovations

What are promising innovations and abstractions for building future high-confidence safety systems?

It is extremely difficult to determine, which specific techniques or technologies are the most innovative or make the best promise, mostly because their suitability and usefulness have to be proved over time and a range of applications. However, a few essential directions in innovation can be mentioned [9]:

1) Improvement of quality and trustworthiness of products and tools via advances in verification and validation, possibly via the application of formal approaches, such as model checking, has been already in a view of researches for some two decades and is still making a promise.
2) Design diversity as an essential technique in improving computer and software safety has been used successfully for years and will remain to be used as one of the most effective safety techniques thus far.
3) Several newer technologies emerged over the recent years, of which we mention only two: model based development and active safety systems.
4) Present authors' own research based on the concepts of a safety shell [10] and Bayesian belief networks [11] has also a potential to improve safety in an array of applications.

It seems that a significant progress to develop new innovative technologies for safety assessment and assurance may not be possible without some major concentrated effort towards funding respective research. This should be an effort similar to steps taken a few years ago to sponsor security research. The scale of funding should be such that development of innovative solutions would be truly possible. For comparison, it is worthwhile mentioning that the European Commission has recently provided over €3M of funding for a joint university-industry project on active system safety [12].

5. Possible Milestones and Conclusion

What are possible milestones for the next 5-to-10 years?

The way the present authors see progress made possible in the next 5-10 years is via a significant coordinated effort of respective government agencies and industrial sectors, driven by the following three factors:

1) Setting priorities in research directions, for example to define and verify measurable safety metrics.
2) Establishing educational preferences to design and implement changes in the computing curricula as well as by offering respective training for safety engineers.
3) Enforcing qualification and certification processes, so that industry would become better aware how their respective products and activities will undergo thorough but transparent assessment.

Certainly, all this requires a significant increase in the level of funding, which may not be possible without decisive legislative actions. It should be made clear to the decision makers that if cost minimization will continue to be an essential factor in safety related industries, then we may soon experience the kind of failures which were caused not so long ago by breaches in security.

Acknowledgments

This project was supported in part by the Aviation Airworthiness Center of Excellence (AACE) under contract DTFA-0301C00048 sponsored by the Federal Aviation Administration (FAA). Findings contained herein are not necessarily those of the FAA. J. Zalewski acknowledges additional support from the Florida Space Grant Consortium under Grant No. UCF01-E000029751

References

[1] Redmill F. (Ed.), *Dependability of Critical Computer Systems*, Vol. 1 & 2, Elsevier Applied Science, London, 1988/89

[2] Leveson N.G., *Safeware – System Safety and Computers*, Addison-Wesley, Reading, Mass., 1995

[3] Kornecki A.J., J. Zalewski, Experimental Evaluation of Software Development Tools for Safety-Critical Real-Time Systems, *Innovations in Systems and Software Engineering – A NASA Journal*, Vol. 1, pp. 176-188, 2005

[4] RTCA, *Software Considerations in Airborne Systems and Equipment Certification*, Report RTCA/DO-178B, Washington, DC, 1992

[5] Kornecki A., J. Zalewski, The Qualification of Software Development Tools from the DO-178B Certification Perspective, *CrossTalk – The Journal of Defense Software Engineering*, Vol. 19, No. 4, pp. 19-22, April 2006

[6] Redwine, Jr., S.T (Ed.), *Secure Software Assurance: A Guide to the Common Body of Knowledge to Produce, Acquire, and Sustain Secure Software*, Draft Version 0.9. U.S. Departments of Homeland Security and Defense, January 2006

[7] Kornecki A., J. Zalewski, J. Sosnowski, D. Trawczynski, A Study on Avionics and Automotive Databus Safety Evaluation, *The Archives of Transport*, Vo. 17, No. 3-4, pp. 107-132, 2005

[8] Software Tools Forum, Embry-Riddle Aeronautical University, Daytona Beach, FL, May 18-19, 2004, URL: http://www.erau.edu/db/campus/softwaretoolsforum.html

[9] Zalewski J., W. Ehrenberger, F. Saglietti, J. Gorski, A. Kornecki, Safety of Computer Control Systems: Challenges and Results in Software Development, *Annual Reviews in Control*, Vol. 27, pp. 23-37, 2003

[10] Sahraoui A.E.K., E. Anderson, J. van Katwijk, J. Zalewski, Formal Specification of a Safety Shell in Real-Time Control Practice, *Proc. 25th IFAC/IFIP Workshop on Real-Time Programming*, Mallorca, Spain, May 15-19, 2000, pp. 117-123

[11] Zalewski J., A.J. Kornecki, H. Pfister, Numerical Assessment of Software Development Tools in Safety-Critical Systems Using Bayesian Belief Networks, *Proc. Int'l Multiconference on Computer Science and Information Technology*, Wisła, Poland, November 6-10, 2006, pp. 433-442.

[12] ONBASS – An Onboard Active Safety System, URL: http://www.onbass.org and http://ec.europa.eu/research/aeronautics/projects/article_3704_en.html

Authors' Bios

Dr. Andrew J. Kornecki is a Professor at the Dept. of Computer and Software Engineering, Embry Riddle Aeronautical University. He has over twenty years of research and teaching experience in areas of real-time computer systems. He contributed to research on intelligent simulation training systems, safety-critical software systems, and served as a visiting researcher with the FAA. He has been conducting industrial training on real-time safety-critical software in medical and aviation industries and for the FAA Certification Services. Recently he has been engaged in work on certification issues and assessment of development tools for real-time safety-critical systems. He is currently, with Dr. Zalewski, conducting a study on tool qualification for complex electronic hardware, sponsored by the FAA.

Dr. Janusz Zalewski is a Professor of Computer Science and Engineering at Florida Gulf Coast University. Before taking a university position, he worked for various nuclear research labs, including the Data Acquisition Group of Superconducting Super Collider and Computer Safety and Reliability Center of Lawrence Livermore National Laboratory. He also worked on projects and consulted for a number of private companies, including Lockheed Martin, Harris, and Boeing. He served as a Chairman of IFIP Working Group 5.4 on Industrial Software Quality and of an IFAC TC on Safety of Computer Control Systems. His major research interests include safety-related real-time computer systems. He currently works with Dr. Kornecki on a study for the FAA on tool qualification for complex electronic hardware.

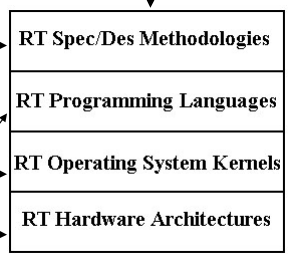
Selected Issues in Computer System Safety

Andrew J. Kornecki
Embry Riddle Aeronautical University
Daytona Beach, FL 32114, USA
kornecka@erau.edu
http://faculty.erau.edu/korn/

Janusz Zalewski
Florida Gulf Coast University
Fort Myers, FL 33965, USA
zalewski@fgcu.edu
http://www.fgcu.edu/zalewski/

Lecture Outline

1) Fundamental Concepts
2) Design Principles & Architectures
3) System Safety
4) Real-Time Programming
5) Databus Safety
6) Case Studies & Summary



System Safety: Dependability

- **Dependability** is the property of the system that justifies reliance on its services

  {more on the topic: "Dependability: Basic Concepts and Terminology", Edited by Laprie, J.-C., Springer Verlag, 1992, ISBN: 3-211-82296-8}

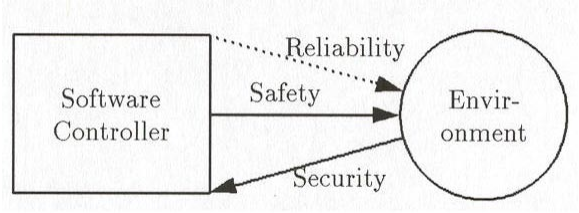
- Dependability is encapsulation of the following properties/abilities *{adapted from Laprie}:*
  - **Reliability** - probability to function correctly over a given period of time
  - **Security** - ability to prevent unauthorized access and system damage
  - **Safety** - ability of not harming people and not cause property damage

System Safety: Dependability

Dependability involves:

- **Attributes** - the metrics for evaluation of system services (safety, reliability, security, availability, integrity, maintainability, confidentiality, etc.)
- **Impairments** - causes or results of lack of dependability (error, fault, failure)
- **Means** - the methods used to deliver dependable services (fault prevention, removal, detection, tolerance)

System Safety: Dependability



Relationships among Reliability, Safety and Security Attributes

System Safety: Dependability

Reliability – failure does not lead to severe consequences (high risk) to the environment or computer system, nevertheless improving the failure rate is of principal concern

Safety – failure leads to severe consequences (high risk) to the environment (and possibly to computer)

Security – failure leads to severe consequences to the computer system (and possibly to the environment)

System Safety: Dependability

Example of dependability issues in a car embedded control software:

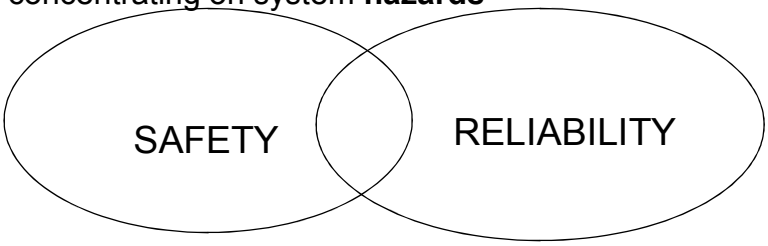
Reliability – ignition control, cruise control, fuel gauge, odometer, etc.

Safety – air bag, seat belts control, anti-lock brakes, etc.

Security – door locks, alarms, etc.

System Safety: Dependability

- **Reliability** involves **bottom-up** activities focusing on system **failures**
- **Safety** involves **top-down** approach concentrating on system **hazards**



fail-safe state defined
(reliability is secondary)

no safety analysis
(reliability assessment only)

System Safety: Basic Terms

- **Safety** is a characteristic of a system ensuring that it will not endanger human life, property or environment
- **Safety-critical software system** is a software intensive system involved in assuring that safety of equipment or plant it is interfacing with is not compromised
- Software Safety is achieved by implementing **features** and **procedures** ensuring that a product performs predictably under normal and abnormal conditions so the likelihood of unplanned events is minimized and their consequences are controlled and contained

System Safety: Basic Terms

- **Hazard** is the capability of the system to harm the people, destroy the property or environment
- Nature of the hazard defines the way how it works and how it can be controlled *(radiation, electric shock, mechanical break)*
- The hazard is a **potential** danger to do harm during the system operation
- The **actual** occurrences of hazards are **incidents** and **accidents**

System Safety: Basic Terms

- The role of safety features and procedures is to ensure safety preventing incidents and accidents
- **Incident** is an occurrence of a situation that could result in a severe consequences (in terms of loss of life or property) but it was prevented or the situation was kept under control
- **Accident** is an unplanned event or series of events that results in death, injury, environmental or material damage
- Both **incidents** and **accidents** are exemplifying **safety violation**

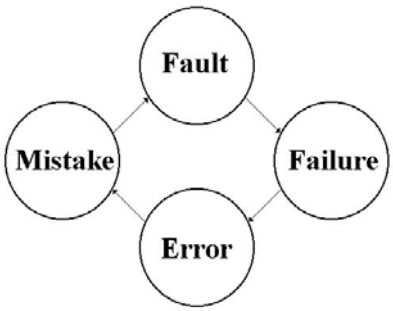
System Safety: Basic Terms

- **Severity** of hazard describes consequences of potential accident (in terms of the human lives or monetary value)
- **Likelihood** of a hazard defines how often can we expect the hazard to occur (in terms of how many times per time unit)
- **Risk** is the combined measure of **severity** and **likelihood** of a hazard – *likelihood of hazard leading to an accident (combined with hazard severity)*

System Safety: Basic Terms

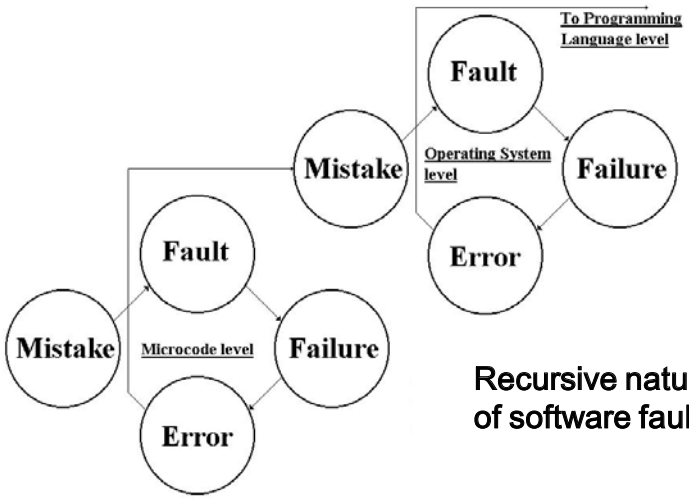
- **Mistakes** are made by people (specification, design, coding, manufacturing, etc.)
- **Fault** is an internal defect within hardware or software caused by a **mistake**, component imperfection, or external disturbance (or inability of a function to perform a required action)
- **Failure** is an external view of the system, showing its inability to perform required functions
- **Error** is the difference between computer (observed, measured) value and the true (specified or theoretically correct) value (it's a manifestation of a **failure**)

System Safety: Basic Terms



Fault propagation cycle

System Safety: Basic Terms



Recursive nature of software faults

System Safety: Basic Terms

- **Mistakes** made by people (*"errare humanum est"*) are the primary reasons that "something went wrong"
- **Failure** is when the system fails to perform its required function in the operational phase
- Failure can be caused by:
  - **user** makes ***how-to-use*** **mistake**
  - **fault** (or **defect**) within the **hardware**
  - **fault** (or **bug**) within the **software**

NOTE: Keep in mind that safety may be compromised when no failure occurs.

System Safety: What went wrong?

- **How-to-use mistake** is more likely to happen because:
  - of **user mistake** during analysis or training
  - the **product** is **imperfect** (too complex, difficult to use, poor diagnostics)
  - there is a **fault within the software**
- **Software bug** (or imperfect product) is due to the **developer's mistake**
- **Hardware defect** is due to:
  - **designer's mistake**
  - **manufacturer's mistake**

System Safety: What went wrong?

- Environmental and operating conditions (disabling interrupts may lead to failing an interrupt driven safety critical function)
- Logic control by Real Time Executive (order of processing may impact the failure conditions)
- System function calls (detailed understanding their operations and side-effects is critical)
- System resources (implicit use of memory in stack ops)
- Timing (deadlines, jitter, or drift may prove dangerous)
- Software architecture (choice of representation may impact safety)

System Safety: What went wrong?

- Most of development methods do not provide guarantee that timing constraints be met, thus verification of timing requirements is carried out after writing the code
- Such approach can be costly because of late fault detection and need to re-write the code for speed
- Defining timing requirements can be ambiguous, thus notations allowing formally analyze or animate the system model are recommended
- Safety requirements must be traceable through the progression of the product artifacts *(requirements => design => code => operation)*

System Safety: Techniques

Safety Techniques basically fall into two broad categories:

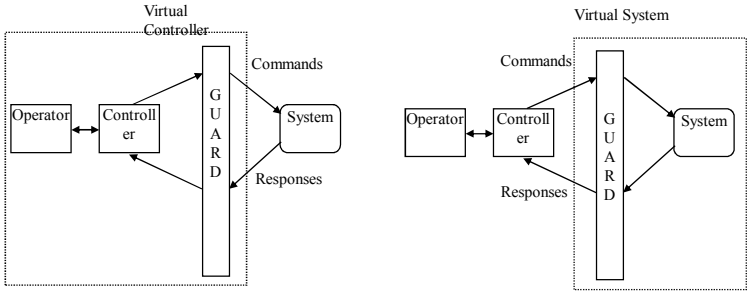
- Design Techniques (to improve the *product*)
- Process Techniques (to improve the *process*)

System Safety: Design Techniques

Major architectural safety techniques:

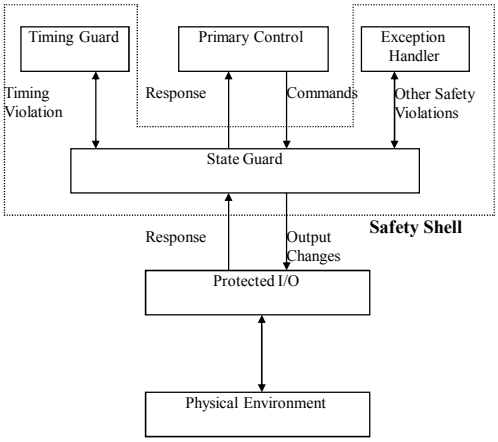
- **redundancy** - using multiple components to carry the same task
- **diversity** - two components (channels, systems) to carry the same task are based on different technologies

System Safety: Design Techniques



Principle of a Safety Shell

System Safety: Design Techniques



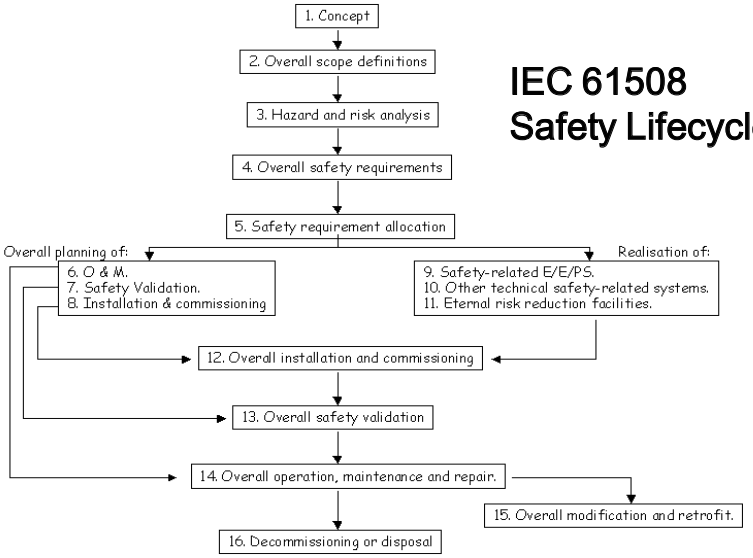
Guards Incorporated into a Safety Shell

System Safety: Process Techniques

Safety Lifecycle according to IEC 61508:

- IEC 61508 is a standard for the life-cycle management of Instrumented Protection Systems – it formalizes a risk-based approach to establishing target Safety Integrity Levels (SIL) and assessing if systems meet these targets
- **IEC 61508**: *"The necessary activities involving safety-related systems, occurring during a period of time that starts at the concept phase of a project and finishes when any safety-related systems are no longer available for use"*

System Safety: Process Techniques



IEC 61508
Safety Lifecycle

System Safety: Process Techniques

- Hazard Operability Analysis (HAZOP)
- Failure Mode and Effect Analysis (FMEA)
- Failure Mode and Effect Criticality Analysis (FMECA)
- Fault Tree Analysis (FTA)
- Event Tree Analysis (ETA)
- Common Mode Failure Analysis (CMF)
- Cause Consequence Diagrams (CCD)
- Petri Nets

Real-Time Programming

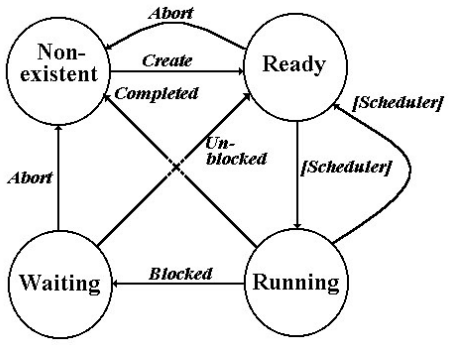
Basic Concepts

- Concurrency and basic program properties
- Programming language features
- Real-Time kernel features
- Timing issues in real-time programs
- Practical aspects of real-time scheduling
- Board Support Packages & Device Drivers

Real-Time Programming: Concurrency

- **TASK -** a unit of concurrency executing sequentially itself, designed to fulfill a specific system function, typically defined by:
  - ***event*** *- environmental or internal stimulus occurring at a time point requiring response*
  - ***activity*** *- a set of operations responding to the event requiring time*
- Task can be implemented as:
  - **PROCESS -** a virtual computing environment set up to run as an application program (contains its own data, code, context, & resources)
  - **THREAD -** a sequence of instructions executed within the context of a process

Real-Time Programming: Concurrency



States of Concurrent Tasks

Real-Time Programming: Concurrency

Concurrency Terms

- Synchronization
- Critical Section
- Mutual Exclusion
- Reentrancy
- Deadlock
- Pre-emption
- Safety and Liveness
- Scheduling

Real-Time Programming: Concurrency

```
task body SLICER is
   ME : ID := GET_ID;
begin
   PUT_LINE("Task " & ID'IMAGE(ME) & " up & running");
   loop
      -- delay 0.0;
      COUNTS(ME) := COUNTS(ME) + 1;
   end loop;
exception
   when others =>
      PUT_LINE("Exception in slicer " & ID'IMAGE(ME));
end SLICER;
```

10-task example of concurrent execution

Real-Time Programming: Concurrency

10-task Slicing Example (null Main)

Case 1: delay 0.0 -- inactive	Case 2: delay 0.0 -- active

`Task 1 up & running`	`Task 1 up & running`
	`Task 2 up & running`
	`...`
	`Task 10 up & running`

Real-Time Programming: Concurrency

Case 3:
delay 0.0 -- inactive
Exceptions active

```
Task 1 up & running
Exception in task 1
...
Task 10 up & runing
Exception in task 10
```

Case 4:
delay 0.0 -- active
Exceptions active

```
Task 1 up & running
...
Task 10 up & running
Exception in task 1
...
Exception in task 10
```

Real-Time Programming: Concurrency

10-task Slicing Example (Main killing children)

Case 5:
delay 0.0 -- inactive
prio Main < prio Tasks'

```
Task 1 up & running
```

Case 6:
delay 0.0 -- inactive
prio Main > prio Tasks'

```
Main waiting 10 sec
Task 1 up & running
Main killing off ch.
Task 1 count is 16M
Task 2 count is 0
...
Task 10 count is 0
```

Real-Time Programming: Concurrency

Case 7:
delay 0.0 -- active
prio Main > prio Tasks'

```
Main waiting 10 sec
Task 1 up & running
...
Task 10 up & running
Main killing off ch.
Task 1 count is 78K
Task 2 count is 78K
...
Task 10 count is 78K
```

Case 8:
delay 0.0 -- active
prio Main < prio Tasks'

```
Task 1 up & running
```

Real-Time Programming: Languages

- An informal scan of the real-time (embedded, dedicated, safety-critical) market reveals:
  - 30% assembly and legacy languages
  - 30% Ada
  - 30% C/C++
  - 10% other (100+ other languages)
- C and Ada are the most commonly used languages in civil aviation today
- C++ is gaining popularity, but its usage is still limited

Real-Time Programming: Languages

FEATURE	Ada	C/C++	Java
Memory Management	automatic	manual	garbage collected
Run-Time Efficiency	high	high	medium
Run-Time Predictability	high*	OS dependent	low
Concurrency Control	language features	OS specific	language library

Real-Time Programming: Languages

Language Safety Features:

- Formally defined syntax; Block structure
- Strong typing; Wild (unstructured) jumps
- Memory overwrites; Memory exhaustion
- Dangling pointers; Variable initialization
- Model of floating-point arithmetic
- Exception handling; Reentrancy
- Separate compilation with cross-checking
- Temporal predictability of loops

Efforts include: SPARK, MISRA-C, PEARL.

Real-Time Programming: Languages

Strong Typing – strict application od data type checking rules to prfevent misuse of variables and data

```
int a:
float x;        // and x are of different types
a = x;          // formally, this is incorrect,
                // but C/C++ may allow it
a = (int)x;     // mode appropriate coding
```

Fortran allows implicit declarations
C/C++ and Java allow implicit type casting
Ada requires explicit type casting

Real-Time Programming: Languages

Exception – an unexpected situation that may cause a program to crash.

Examples: division by 0, overflow, reference to nonexisting object (memory, device), I/O error, etc.

Exception handling – to provide facilities within a language to neutralize consequences of exceptions.

Real-Time Programming: Languages

Usual sequence of actions in exception handling:

1. Exception handler included in a program.
2. Exception raised during program execution.
3. Control is transferred to exception handler.
4. Handler executes and exits to the surrounding block.

```
x := a/b;    -- What if b=0?
…
exception
   when CONSTRAINT_ERROR => x=MaxInt;
end;
```

Real-Time Programming: Languages

Unstructured jump (wild jump) – a program jump which is not controlled by the programmer.

Unstructured jumps are most likely to occur in case statement or their equivalents.
Examples include:
- incomplete coverage of cases (missing default/others)
- erroneous exit from cases (missing break)
- incomplete if/else pairs.

Real-Time Programming: Languages

C/C++ pair setjpm() and longjmp():

```
int setjump(jmp_buf env)
// Saves state info in env for use by longjmp
void longjmp(jmp_buf env, int v)
// Restores the state saved by setjmp
```

Real-Time Programming: Languages

Memory overwrite – an uncontrolled access to arbitrary memory locaations.

May be cause by: erroneous pointers, out-of-bounds array indexes, dynamic allocation.

The programmer must remember that a pointer is not just an address; it is an address of a data item of a certain type.

Real-Time Programming: Languages

```
// Check memory space available
int * arr , j=0;
for ( ;  ; ) {
    j++;
    arr = (int *)malloc(TEN_K);
    printf("%d ", j);
}
```

Real-Time Programming: Languages

Reentrancy – subprogram property that allows it to be executed by multiple callers at the same time.

Need for reentrancy is typical in multithreaded programs. Therefore library routines are usually indicated MT-safe or not.

Real-Time Programming: Languages

Other language aspects:

- variable initialization
- order of evaluation vs. operator precedence
- spawning processes via fork
- killing concurrent units.

Real-Time Programming: Languages

```
// Both should be avoided
x = i++ + a[i];
x = (i++) + a[i];

// What is the result, and why?
int i = 0;
i = i+++i;
```

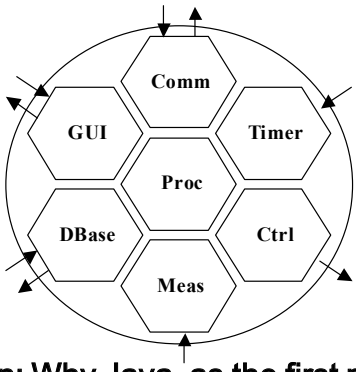
Real-Time Programming: Languages

```
// When if has both branches
// executed simultaneously!
if (fork()) {
    …  // some code
}
else {
    …  // other code
}
```

Real-Time Programming: Languages

Beware of problems with destroying concurrent units:
- Ada tasks via abort
- Aunix processes via kill()
- threads vi acancellation.

Real-Time Programming: Languages



Observation: Why Java, as the first programming language in common use, included GUI and Networking as part of the language? Do LabVIEW and MATLAB show similar trend?

Real-Time Programming: RT Kernel

Concept of RTOS/Kernel Operation:

- Strong distinction between internal system operations and the user tasks
- RTOS kernel does not participate in the priority scheme - it operates in the **hardware context**
- Peripheral interrupts handled by extensions to the kernel (device drivers) which also function outside normal application task prioritization

Real-Time Programming: RT Kernel

Concept of RTOS/Kernel Operation:

- User tasks communicate with the kernel and perform most I/O through entry points or calls into the drivers - I/O is processed outside the user application context
- Modern RTOS uses threaded micro-kernel with fast response and options for handling interrupts at the system priority level

Real-Time Programming: RT Kernel

Basic Terminology

- **EVENT** – a result of an externally or internally generated occurrence handled by the processor
- **LATENCY** - time required to recognize and start responding to an event
- **RESPONSE TIME** - time interval between presentation of an input *(stimulus)* and the appearance of the associated output *(response)*
- **DEADLINE** - a time point before which a specific event must occur (e.g. the task must complete the execution)

Real-Time Programming: RT Kernel

Basic Terminology

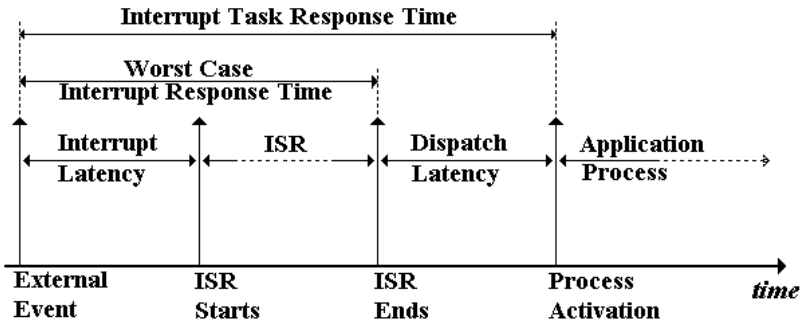
- **INTERRUPT LATENCY** – the time interval between the occurrence of an external event and the start of the first instruction of the interrupt service routine
- **INTERRUPT LATENCY INVOLVES**: hardware logic processing, interrupt disable time, handling higher hardware priority interrupts, switching to handler code (saves, etc.)

Real-Time Programming: RT Kernel

Basic Terminology

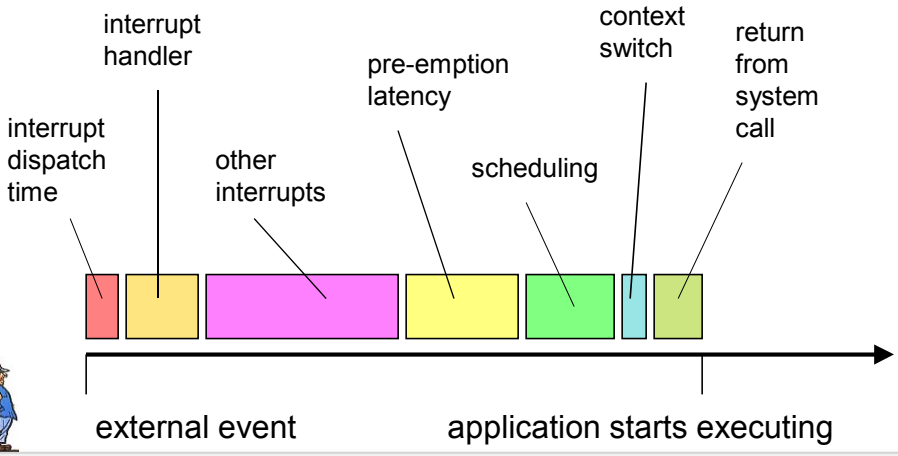
- **DISPATCH LATENCY** – the time interval between the end of interrupt handler code and the first instruction of the process activated (made runnable) by this interrupt.
- **DISPATCH LATENCY INVOLVES**: OS decision time to reschedule (non-preemptive kernel state), context switch time, return from system call.

Real-Time Programming: RT Kernel



Real-Time Programming: RT Kernel

Contributions to Interrupt Task Response Time



Real-Time Programming: RT Kernel

Kernel Responsiveness Involves:

- **INTERRUPT LATENCY**
- **TASK DISPATCH LATENCY**
- **(WORST CASE) INTERRUPT RESPONSE TIME**
  (Interrupt Latency + Worst case Execution of the Interrupt Handler + Interrupt Exit Overhead)
- **INTERRUPT TASK RESPONSE TIME**
  (Interrupt Response Time + Dispatch Latency)

Real-Time Programming: RT Kernel

Schedulability and Determinism

- **SCHEDULABILITY** - a property of a set of tasks ensuring that all tasks will meet their respective deadlines
- **PREDICTABILITY** - the property of meeting the temporal determinism criteria
- **TEMPORAL DETERMINISM -** the situation in which timing properties of the system are known (or bounded) for each set of inputs

Real-Time Programming: RT Kernel

Topics important but not covered here:

- **Real-Time Scheduling**

  „What Every Engineer Needs to Know about Rate-Monotonic Scheduling"

  IN: Advanced Multimicroprocessor Bus Architectures, IEEE Computer Society Press, 1995, pp. 321-335,

  and Real-Time Magazine, Issue 1/95, pp. 6-24

- **Device Drivers**

  „Teaching Device Drivers Technology in a Real-Time Systems Curriculum"

  IN: Real-Time Systems Education III, IEEE Computer Society Press, 1999, pp. 42-48

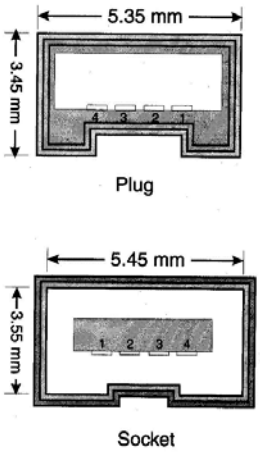
  and at http://www.wrs.com/univ/html/featurevol4.html

Databus Safety

Databus Characteristics

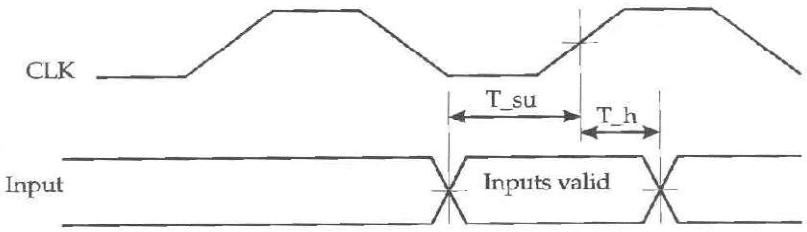
- **Mechanical** properties concern bus wiring, connectors, their pinout, and module design and dimensions
- **Electrical** (or optical) properties are related to signal levels and their dynamics to carry information, including electromagnetic characteristics
- **Logical** properties concern the protocol of exchanging information over a bus

Databus Safety



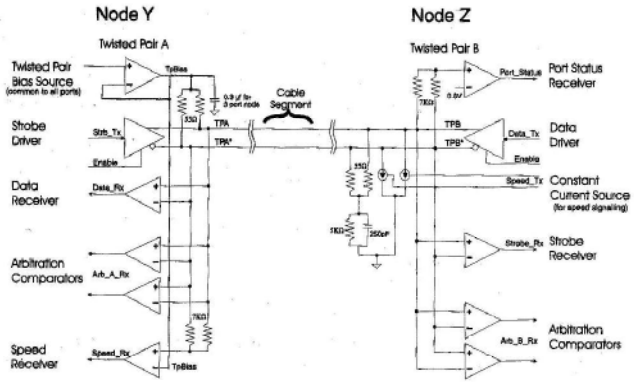
Example of mechanical properties of the connector for FireWire bus.

Databus Safety



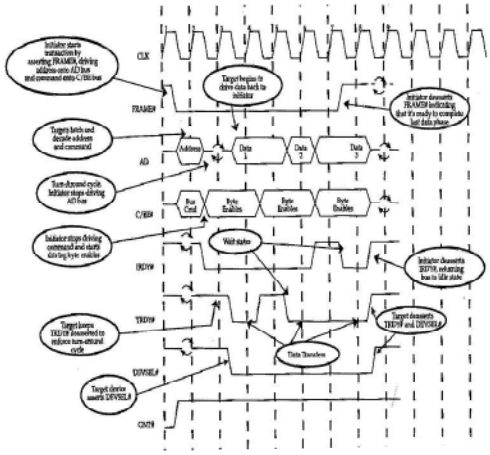
Example of electrical properties of the PCI bus input signals (T_su – setup time, 7-12 ns; T_h – hold time).

Databus Safety



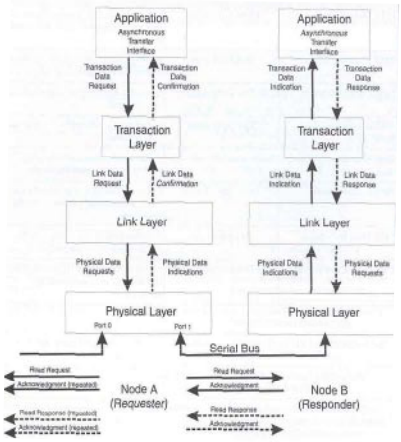
Electrical interface between two FireWire nodes.

Databus Safety



Example of electrical properties and low-level bus protocol for the PCI bus Read Transaction.

Databus Safety



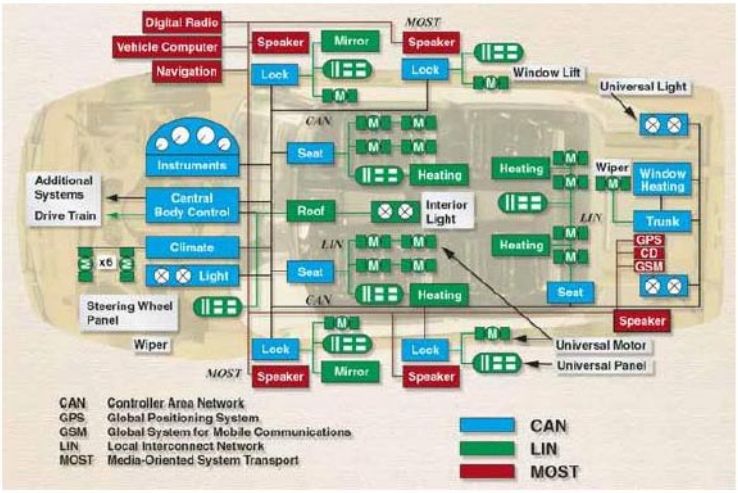
Example of logical properties of the bus for FireWire Asynchronous READ Transaction.

Databus Safety

Specifics of the Bus Protocol:

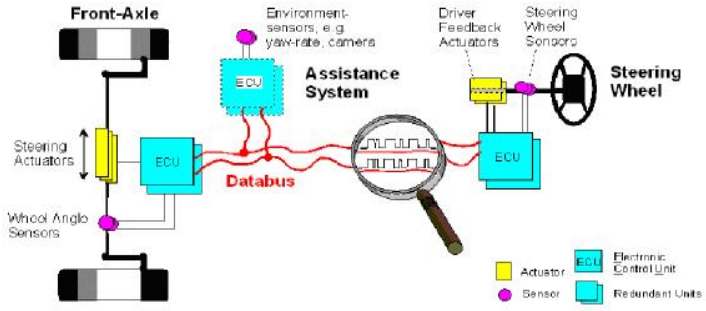
- ***Bus arbitration***
  competing for bus access
- ***Data transfer***
  how devices exchange data once they obtain bus access
- ***Fault handling***
  dealing with bus errors

Databus Safety



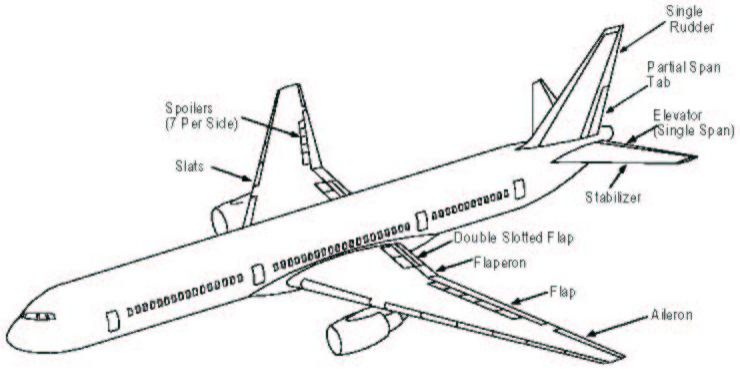
An Example of Modern Vehicle Network (Leen 2002)

Databus Safety



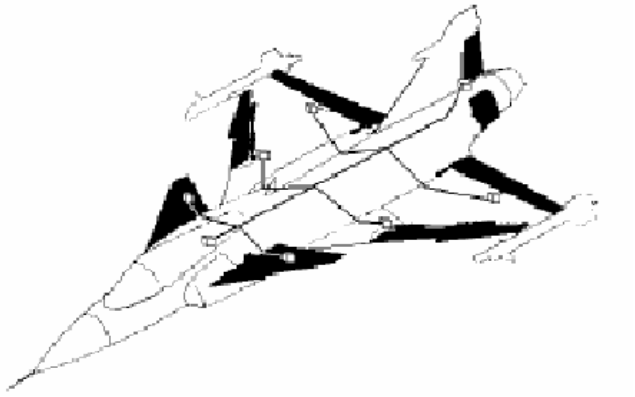
Steer-by-Wire System (Waern 2003)

Databus Safetv



Distributed Flight Control System for Boeing 777 Aircraft

Databus Safety



Distributed Flight Control System for JAS 39 Gripen Aircraft (Johansson 2003)

Databus Safety

Databus	Type	Architecture	Medium	Rate	Encoding
Arinc 429	serial unidir.	single master	2 wires	100kb/s	RTZ bipolar
MIL1553	serial bi-dir.	single master	twist pairs	1 Mb/s	biphase Manch.
Arinc 629	serial bi-dir.	multi master	twist pairs	2 Mb/s	Manchester II
Arinc 659	serial bi-dir.	quad redund	twist pairs	30MHz	biphase Manch.
FlexRay	serial bi-dir.	fault-tolerant	optic/wire	10Mb/s	undefined
CAN	serial bi-dir.	multi-master	twist pairs	1 Mb/s	NRZ + bit stuff
TTP/C	serial bi-dir.	dbl redund	twist pairs	25Mb/s	MFM
IEEE1394	serial	d-chain/tree	twist pairs	400Mb/s	LVDS
Safe-Wire	serial bi-dir.	master-slave	twist pairs	200 kb/s	3-level
SpaceWire	serial bi-dir.	master-slave	2 wires	> 2Mb/s	undefined

Databus Safety

Risk Assessment Process

1) Multicriteria-based Safety Assessment
2) Hazard Analysis
3) Failure Mode Analysis

Databus Safety

Criterion	Selected Evaluation Factors
Safety	Availability and reliability; Partitioning; Failure detection; Common cause/mode failures; Bus expansion strategy; Reconfigurability; Redundancy management
Data Integrity	Maximum error rate; Error recovery; Load analysis; Bus capacity; Security
Performance	Operating speed; Schedulability of messages; System interoperability; Bus length and max. load; Retry capability; Bandwidth; Data latency; Transmission overheads
EMC	Switching speed; Pulse rise and fall times; Wiring; Shielding effectiveness; Lightning/radiation immunity
Design Assur.	Compliance with standards (such as DO-254/DO-178B)
V&V	Examples: functionality testing, system testing, failure management, degraded mode operation
Configuration Management	Examples: change control, compliance with standards, documentation, interface control, system analysis, etc.
Continued Airworthiness	Lifetime issues, such as physical degradation, in-service modifications and repairs, impact analysis. (Rierson/Lewis, 2003)

Databus Safety

Failure Mode	Description
Invalid Messages	Messages sent across the bus Contain invalid data
Non-Responsive	An anticipated response to a message does not occur or return in time
Babbling	Communication among nodes Is blocked or interrupted by uncontrolled data Stream
Conflict of Node Adrs	More than one node has the same identification (Debouk et al. , 2003)

Databus Safety

Potential Hazard	Possible Mitigation
Loss of Power	Dual power system (including battery, wires and connectors)
Loss of Communicat'n	Dual communication system
Loss of Steering	Backup system; Reduced functionality Redundant system; Steer by braking active safety system
Loss of Braking	Backup system; Reduced functionality redundant System; Brake by steering active safety system
Loss of Electronic Throttle	Backup system; Reduced functionality redundant system
Loss of Actuators	Backup actuators; Red. performance actuator
Loss of Sensors (recording driver cmds)	Backup sensors; Red. performance sensor (Chau et al. , 2003)

Databus Safety

Bus Experiments

- Plain simulation for well developed databus networked configurations VME/Raceway
- Actual data transfer experiments with a modern bus FireWire
- Simulation and real experiments for routing in Bluetooth
- Improving Real-Time Characteristic of the Ethernet

Databus Safety

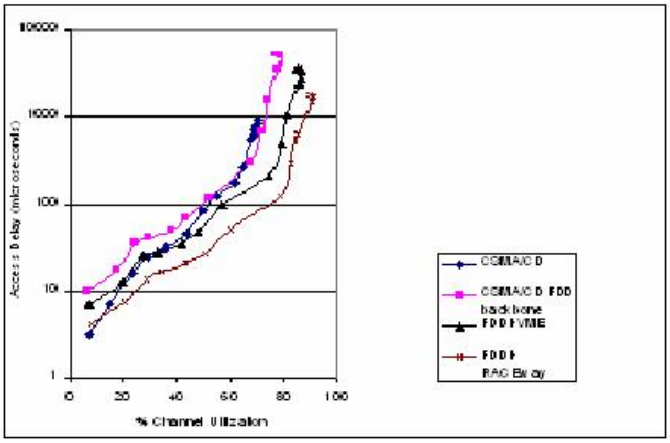
Bus Parameters

- Bus response – access delay vs. bus load
- Bus throughput - data transfer rate vs. packet size
- Interconnect formation and routing
- Predictability of packet transmission time

Databus Safety

Access delay vs. bus load:
When bus load increases,
how does it impact access delay?

Databus Safety

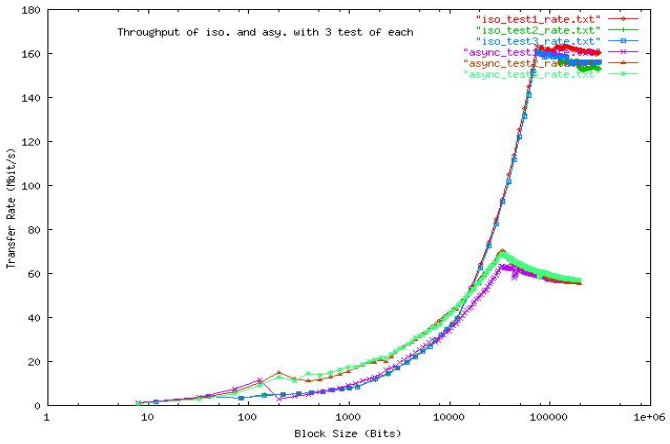


Server Access Delay for 64B Packets

Databus Safety

<u>Bus throughput:</u>
When packet size increases,
how does it impact transfer rate?

Databus Safety

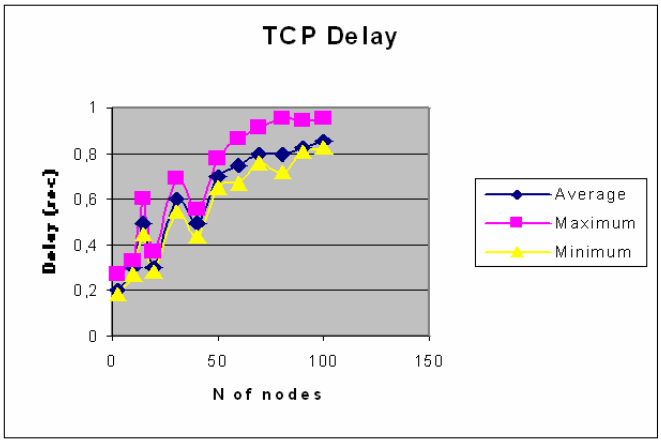


IEEE 1394 Throughput over a Raw Driver
for Asynchronous and Isochronous Modes

Databus Safety

<u>Interconnect formation and routing:</u>
When nodes are being added,
how does it impact access delay
and data transfer rate?

Databus Safety



Bluetooth TCP Delay for Increasing Number of Nodes

Databus Safety

Deterministic Ethernet:

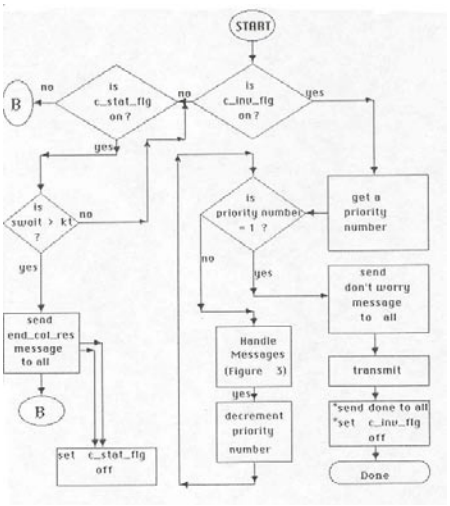
Can Ethernet be made predictable without modification of its CSMA/CD protocol?

Each node is assigned a priority and two flags:

- collision status flag, c_stat_flag (collision resolution in progress)

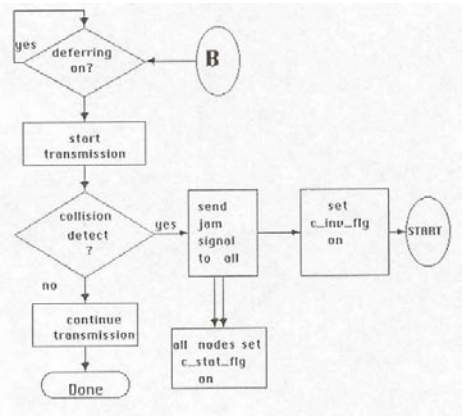
- collision involved flag, c_inv_flag (node was involved in the collision)

Databus Safety



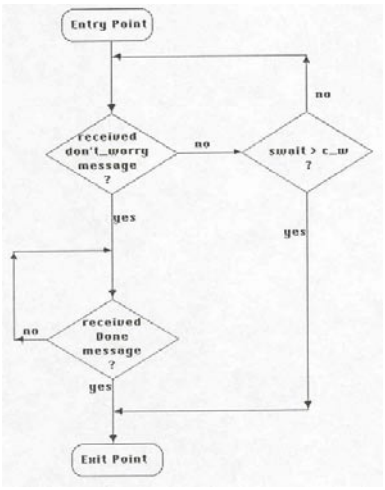
Principle of a deterministic Ethernet protocol.

Databus Safety



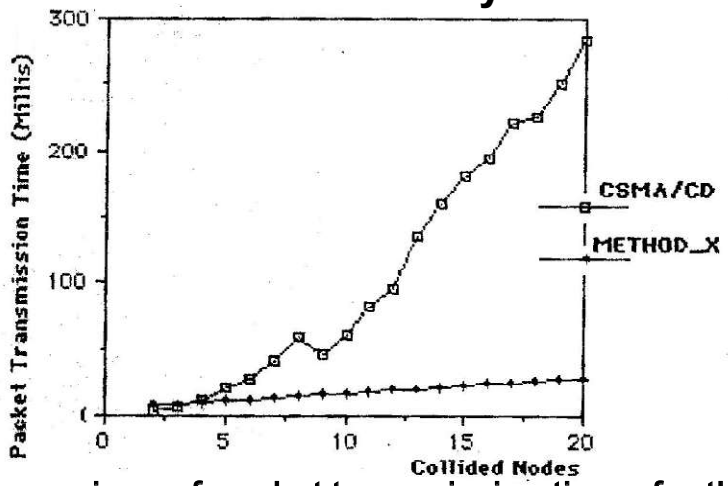
Behavior of a regular CSMA/CD node.

Databus Safety



Handling messages block of the protocol.

Databus Safety



Comparison of packet transmission times for the classic CSMA/CD and the extended protocol.

Databus Safety

Databus Safety

- New area with ongoing research
- Risk assessment methods essential as a starting point
- Definition of critical parameters
- Experimentation needed

OK! Let's go for a beer!!!

Subject Introduction

Archibald McKinlay
Naval Ordnance Safety and Security Activity
Indian Head, MD

I. Introduction – Transforming Systems Safety Performance

This introduction is for three related papers which require a similar background introduction. It is intended that the reader review this introduction before any of the attached papers:

- Hooking into Systems Engineering
- Systems Safety Engineering HR
- Systems Safety in new Architecture and Technologies

Likewise, the reader is spared the redundancy of re-reading the introduction in each paper by reading this header paper.

II. Gap Analyses

Nothing can be fixed without first investigating what is broken.
The first issue at hand in any Gap Analysis is to determine exactly what focal point will be used to center the context. "Begin with the end in mind", Steven Covey in his 7 Habits book, demands we consider what we want out of this gap analysis.
We need first to find the definition of Systems Safety that we are trying to "save".
Systems Safety Engineering has had noble roots in Systems Engineering. Over time, partly because of its success and partly because others could not defend their discipline nor find an established parent, Systems Safety Engineering has both endured law and regulation, de-standardization, and the various Lean 6 sigma and other process improvements. Along the way, even inherited at least two disciplines.

a. First gap is which Systems Safety?

A review of MIL-STD-882D, DoD Standard Practice for Systems Safety, of 10FEB2000, reveals there are actually three parts: systems safety, occupational safety, and environmental safety.

i. Three Safety Disciplines

1. Systems Safety Engineering

System Safety Engineering is an engineering science, derived from Systems Engineering, which is throughout the entire lifecycle of a program. The practice involves risk assessments (better known as hazard assessments) from initial conceptual design, through implementation, test, release, and sustainment. There are few college level engineering courses covering this expertise that extend beyond a few weeks in class. For a while it was possible to have experienced Systems Safety Engineers teach engineering new hires that discipline on-the-job, but with the need to increase billability per employee and cutbacks on resources combined with agile resourcing this is no longer possible.

a. Software Systems Safety Engineering

Software Systems Safety Engineering was added when the software was recognized to present a safety risk. Rather than re-apply the existing Systems Safety Engineer to

comprehend and assess the increasing complexity and size, industry and government created this subset discipline. An augmentation set of courses extended Systems Safety first. Now technology is changing both Systems Safety and Software Safety. An attached paper addresses this problem. For a while it was possible to hire software and digital electronics engineers and have experienced Systems Safety Engineers teach them that discipline on-the-job, but with the need to increase billability per employee and cutbacks on resources combined with agile resourcing this is no longer possible.

2. Occupational Health and Safety

The safety of workers, military, and other humans was already covered by the Systems Safety approach but was not visible to management and did not appear complete to external reviewers. It was also found that educational requirements were very different than the engineering-basis usually required for Systems safety, that is, they were more behavioral versus physics or software. This led to a compromise, rather than re-train all the System Safety Engineers, a compliance-based approach was borrowed from successful OSHA programs and levied through the same MIL-STD-882 document. The strength was the simplistic process. The weakness that remained was that the military environment, or any site-specific environment for that matter, requires in-depth experiential knowledge that is difficult to train in the short timeframe (less than five to ten years). Therefore occupational safety was easier taught to ex-military than the other way around. Long term accomplishment was on-the-job training for new employees with experienced employees providing guidance and mentoring, but with the need to increase billability per employee and cutbacks on resources combined with agile resourcing this is no longer possible.

3. Environmental Engineering and Safety

The Environmental Safety requirement was added in manner similar to the OSHA-style in that it became necessary to make environmental compliance more visible. Again MIL-STD-882 was seen as an easy compromise document in which to add the requirement. Likewise again the education required was different and still again the experiential knowledge was so site-specific that even today the practitioners find it easier to take someone who has already worked in that area (flight-line, ship, sub, explosive ordnance disposal, rocket fuel plant) and then send them to the compliance-based classes rather than the other way around. Indeed it is safer for the practitioner to have the field knowledge first (knows when to duck so to speak). This is still an area where the Systems Safety Engineer is usually teamed with an Environmental Engineer or specialist rather than doing it alone.

b. Second gap is getting connected (to Systems Engineering)

Unlike Systems Safety Engineering, taking MIL-STD-882 as measurable response to a forcing function on that discipline, little has been done to incorporate software requirements and risk management into Systems Engineering. There needs to be a holistic systems integration approach to the updating of Systems Engineering to re-integrate Systems Safety Engineering, Systems Assurance and Security and, of course, Software Engineering. Currently Programs move smartly through a Systems Hardware Critical Design Review (CDR) and yet software risks from the previous Preliminary

Design Reviews of hardware and software are not integrated into a meaningful combined or somehow dependent risk. Several Program hardware systems have been found to ignore software altogether until System Integration or Test phase. There are currently efforts such as IEEE 15288, but it already lacks requirements and interface management sections as it goes to its latest revision. The DoD has efforts in-work to update the Systems Engineering Plan (SEP) but it will be a year more before it is finished.

c. Third gap is too connected (architecture) and too new (FPGA and technologies)

As is pointed out in previous "upgrades" to MIL-STD-882 each added discipline required a change in the typical engineer's abilities, education and experience. The advancing technologies must be viewed in the same model. When a (so-far) safe system is taken and simply attached to the internet for monitoring the safety risk changes, and changes in ways that are not obvious to the traditional Software Safety or Systems Safety Engineer. Just as Moore's Law predicts the increase in microprocessor throughput, so also do the training and education requirements increase.

Technology makes this time really different however. This time the change is at both ends of size: both big and very small architecture. At the big end, the global internet increases potential causes astronomically and simultaneously allows the safety critical functions to be physically spread throughout the system. While at the small end an FPGA or ASIC now changes the ability to see or represent an electrical schematic and simultaneously allows the safety critical functions to be physically spread throughout the system. These technologies are breaking the toolkit of most Systems safety and Software Safety Engineers. Very few architectural-level and FPGA-level failure analyses are done, let alone available or understandable in a general engineering way. Often the technology is proprietary. The technologies are changing so fast that systems are being built right now without the updated training, education, or toolkit being available because neither the chip nor the interface existed at the project's start.

III. Change Evangelism

Change is simple on paper but takes a dedicated evangelist in real life. The evangelist must range wide and far and out-spoken everywhere.

a. Systems Engineering

i. Risk Management

Systems Engineering must return to the roots of risk management and use that to maintain focus in prioritizing tasks in all schedules, meetings and budgets.

ii. More Systems Stuff

Like Systems Safety was made to absorb occupational and then environmental tasks, so also must Systems Engineering reconnect to its many children. All children must coordinate through and with Systems Engineering. This can be done with the Systems Engineering Plan (SEP) and the Integrated Master Schedule (IMS) first.

b. System Assurance

i. Systems, Software, Information and Security

The architecture of the internet and changing world politics has coupled with technology to present new, diverse and complex problems for Security. There was an IT discipline which attempts to make systems secure. Information Systems Security Engineering (ISSE) is the art and science of discovering users' information protection needs and then designing and making information systems to safely resist the forces to which they may be subjected (their definition, and it includes "art"). The dilemma is that most of the methods and techniques were applied before and after, and modifications are made at great cost. The front-end bit is simplified to determining threats and defenses. The center bit, called Engineering-in-Depth (Eid), is actually already performed by Systems Engineers, Software Engineers, Systems Safety and Software Safety Engineers. The end bit, certification and accreditation, looks for "patterns" (similar to your virus detector) and dead code.
The shortfalls in the IT and ISSE world are manifold: they are equally burdened by technology advancement, their best techniques are still manual, and they have few education and training opportunities or courseware that is specific to their domain or application. IT is also not real-time and is often not safety-critical whereas a weapons system is both. The natural progression then is to leverage the integration into Systems Engineering and to also leverage the Systems Safety and Software Safety requirements, analyses, and tests.

c. End Game; initial state - get in front end
    i. Focus on Systems Safety Engineering (not occupational nor environmental) and strengthen the discipline
    ii. Follow Software Engineering integration with Systems Engineering efforts;
    iii. Courses in architecture and new technology failure analyses
d. End Game; final state
    i. Guidebook for Engineers
    ii. S/W acquisition management Best Practices in Contract Language for integrated Systems/Software Engineering, Software Development Plan (SDP), Architectural views, new technology "locations" and functions
    iii. DAU and University courseware and degree programs
    iv. end to end across life-cycle, including sustainment and Intellectual Property (IP) rights.

Transforming Systems Safety and Software Safety Today for the Systems of Systems of Tomorrow

Archibald "arch" McKinlay
Naval Ordnance Safety and Security Activity (NOSSA)
N32, Systems Software Safety Engineering
Indian Head, MD 20640-5151

Providing Ordnance Safety for our Warfighters

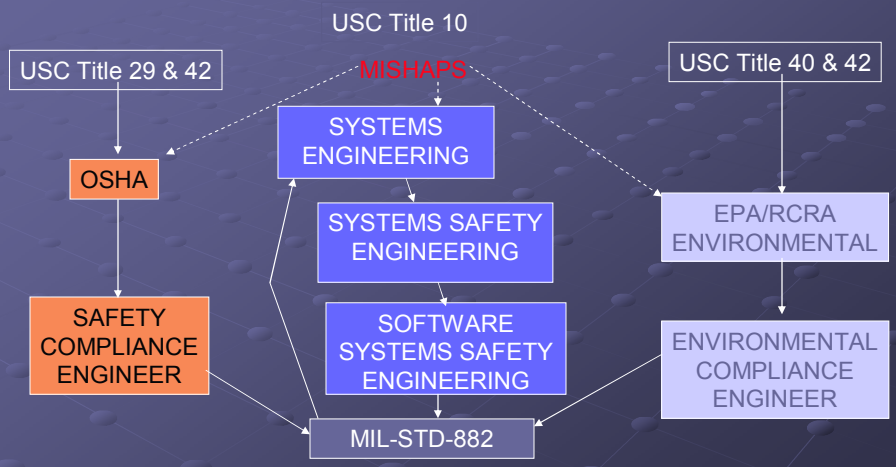
1

First, what do I know?

- Programs are getting "larger"
  - Software size and complexity
  - Number of people assigned
- Systems Engineering is disconnected from Software Engineering
- Processes not standard nor metric'ed
- Subject Matter Experts (SMEs) at "middle" level are few, number and quality varies with domain

2

What in the Sw Systems Safety Tree am I Talking About?



- Systems Sw Safety Engineering is not OSHA and not Environmental but Supports their Risk Mitigation

Why is achieving Minimum Safety Risk so hard and take so long?

- System Safety Engineering requires more time and resources because of the additional contexts to be considered.

Data	*boolean*	***Time***	*boolean*	*Requirements Responsibilty*
RIGHT	1	RIGHT	1	Sys Eng
RIGHT	1	WRONG	0	Safety
WRONG	0	RIGHT	1	Safety
WRONG	0	WRONG	0	Safety

Systems Engr. wants RIGHT Outcome at RIGHT time, but the RIGHT Outcome at the WRONG time can kill
(ComAir into trees example: RIGHT mode WRONG time)

4

How can we improve Systems and Software Engineering Partnerships?

PARTNERS	OBSTACLES	NEEDS
– Defense Safety Oversight Council	• Primarily Focused on OSHA-type Safety (for example, auto accidents)	• Expand to Mitigation thru Design
– Joint Systems Safety and Software Safety		
• Urgent Needs →	• Excuse to Bypass Processes	PM/PFS Team!
• S&T →	• Cost vs. Expertise	Tailoring.
• DDG1000 →	• Technology ahead of Analyses	{Education,
• LCS (both) →	• (same)	Experience,
• Unmanned Systems	• (same)	Research}
• JCIDS →	• Contracting negotiates out	{Adequate Funds & Direction}

5

Plenty of Direction closing in on Software Safety

- Defense Science Board 2000 Report on Defense Software
  - Need to improve processes
- DoD Hon. Mr. Rumsfeld's declaration Memo of Warfighter Safety, Accidents and Mishaps
  - Reduce mishaps>thru Design!
- ASN(RDA) Software Process Improvement
  - focusing on CMMI equivalency
  - No mention of Software Safety
- OSD(AT&L) Software & Systems Engineering Initiative
  - No mention of Software Safety
- USD(AT&L) Memo solidifying changes to JCIDS, 5000.2, 6055.7

DSB "Defense Software" 2000

Rumsfeld "reduce preventable accidents" 19MAY0

USD (AT&L) "Def Acq. System Safety 23SEP04

DODDir 4715.1E ESOH 19MAR05

Krieg "reducing preventable accidents" JCIDS, DODI 5000.2, & DODI 6055.7; 21NOV06

ASN(RDA) Sw Process Improvement 15MAY06

Rumsfeld "progress on reducing preventable accidents?"

OSD(AT&L) Sw & Systems Process Initiative DATE?>

Direction: "We can no longer consider safety as "nice to have""
Follow-on Themes: No Emphasis on Software Safety

The Path we're set upon

- Mr. Krieg directing changes to DODI 5000.2, DODI 5600.72, and JCIDS Process (not strongly linked to contracts)
- Mr. Schaeffer directing changes to make Joint reviews out of Service reviews (sphere of influence)
- Dr Etter's 5 Software Productivity Improvement Initiatives:
  - System Software Engineering
    - New curriculum, USC, UMD
  - Business Impacts
    - No or limited new resources
    - Limited training dollars available
    - Cannot hire/fire needed/finished skill
  - Human Resources
    - Role-based/Right-fit
  - Software Acquisition Management
    - IEEE 12207 as baseline
      - No requirements management
      - No endurance test
      - No systems integration
  - Software Development Techniques

7

Gap Analysis of the "As-Is"

- Inconsistent compliance enforcement
- Inconsistent software measurements and metrics
- Independent Reviews are weak and unfocused
- PEO/PM software process improvement…not happening
- Contracting methods often counter-improvement
- Lack of "bench" in systems software talent
- "Bench" not layered for bigger programs

Providing Ordnance Safety for our Warfighters

8

Filling the Gaps

Need:	Immediate fix:	Long goal:
Compliance	AT&L memos	Reviews, Milestones, metrics
Metrics	Core	safety –centric
Reviews	Major, technical	Working groups
SW Process Improvement	Integrate into SE	Mutually supportive
Contracting	SDP required	SE<>SW<>SSP
Training	Short courses	Degrees (architect)
Experience	Assignments	Career paths

9

Gap Filling doesn't reach the Engineer's daily life!

- The Short and Long term goals are too high a level for the working groups
- Need to view gaps at the work flow level
- Like other Architectures, more views and detail are required
- Stakeholders are: PEO, PM, Chief Systems Engineer, Systems Safety Manager, Systems and Software Safety Engineers and Analysts
- Each has different: Ability, Training, Experience

10

SoS Competency

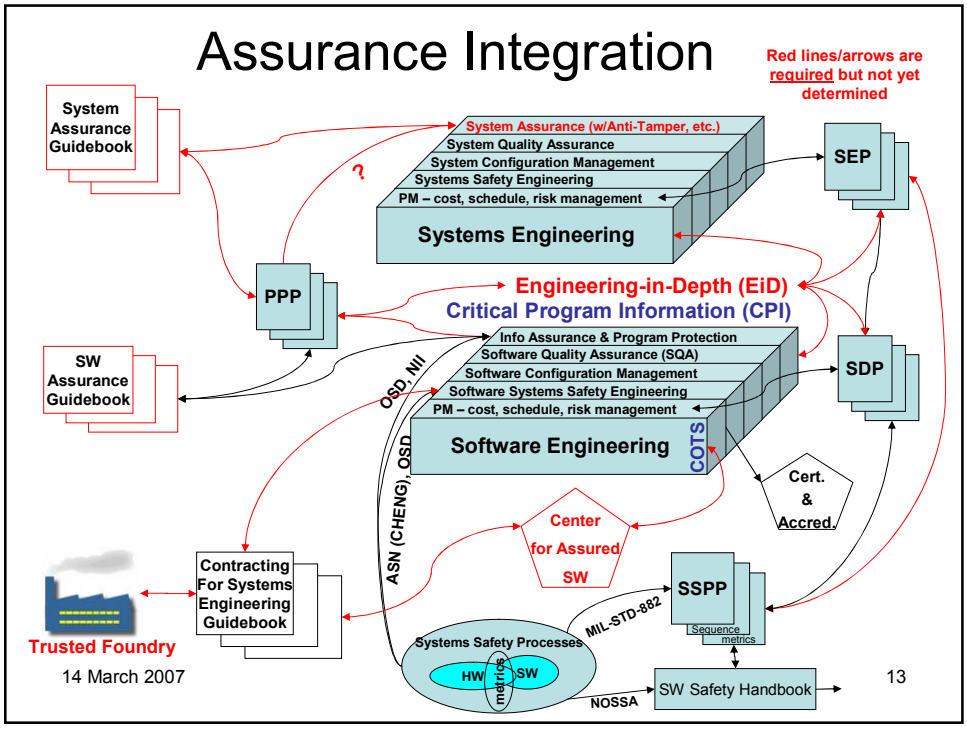
- Support: System-level assurance achieved first
- Ability: “Really big picture” but somehow with details always in mind
- Training: Managing interfaces, many big and many small but with variable importance to the mission/safety
- Experience: systems, systems integration, large numbers of complex interfaces

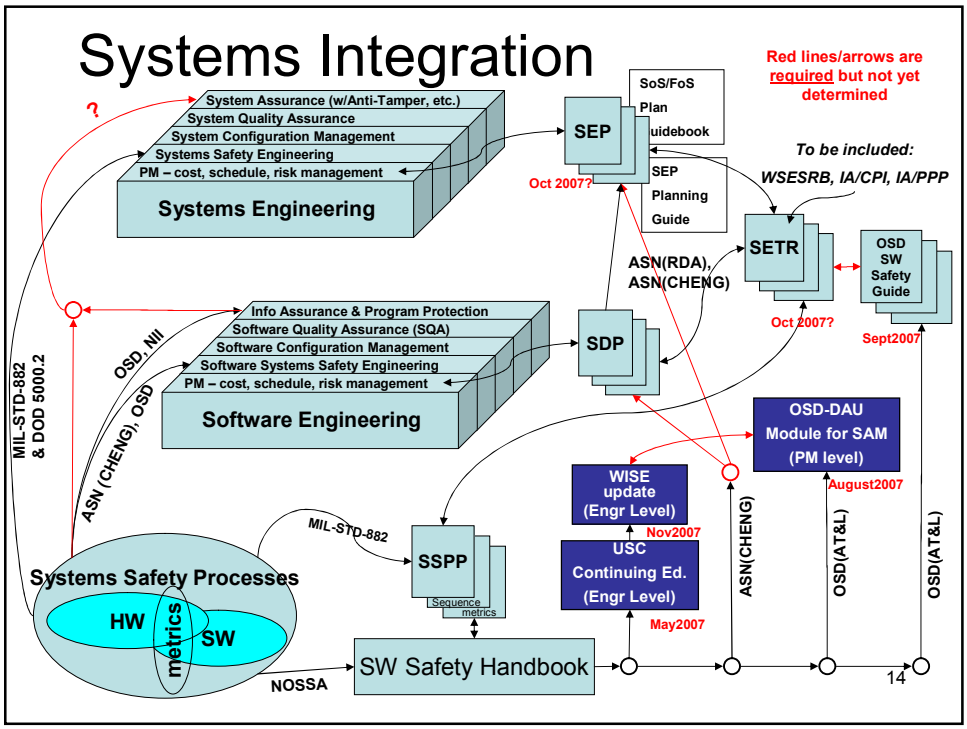
11

SoS Unique Issues

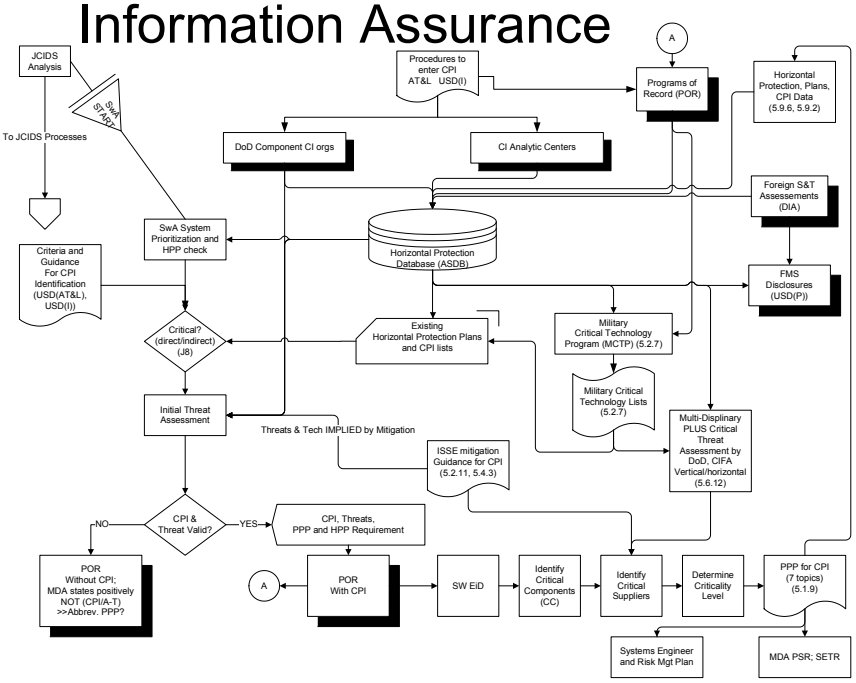
- Agile work and workforce
  - “live to work” alongside “work to live”
- Multiple interfaces and description documents, some internal data flows too
  - Some obvious and some buried details, but different amounts on every system in the SoS (tool to “see”, or experience?)
- Multiple time domains
  - Fourth dimensional complexity (course in timing variable?)
- Multiple States & Modes
  - Too many together (need tool or vision from training)
  - Fifth dimension
- Travel and online collaboration drives team composition to: communicator, admin, tech
  - Need HR to screen or “bin” abilities for selection

12


Assurance Integration
Red lines/arrows are required but not yet determined
System Assurance Guidebook
System Assurance (w/Anti-Tamper, etc.)
System Quality Assurance
System Configuration Management
Systems Safety Engineering
PM – cost, schedule, risk management
Systems Engineering
SEP
PPP
Engineering-in-Depth (EiD)
Critical Program Information (CPI)
SW Assurance Guidebook
Info Assurance & Program Protection
Software Quality Assurance (SQA)
Software Configuration Management
Software Systems Safety Engineering
PM – cost, schedule, risk management
Software Engineering
COTS
SDP
OSD, NII
ASN (CHENG), OSD
Cert. & Accred.
Center for Assured SW
Contracting For Systems Engineering Guidebook
Trusted Foundry
SSPP
Sequence metrics
MIL-STD-882
Systems Safety Processes
HW
SW
metrics
NOSSA
SW Safety Handbook
14 March 2007
13

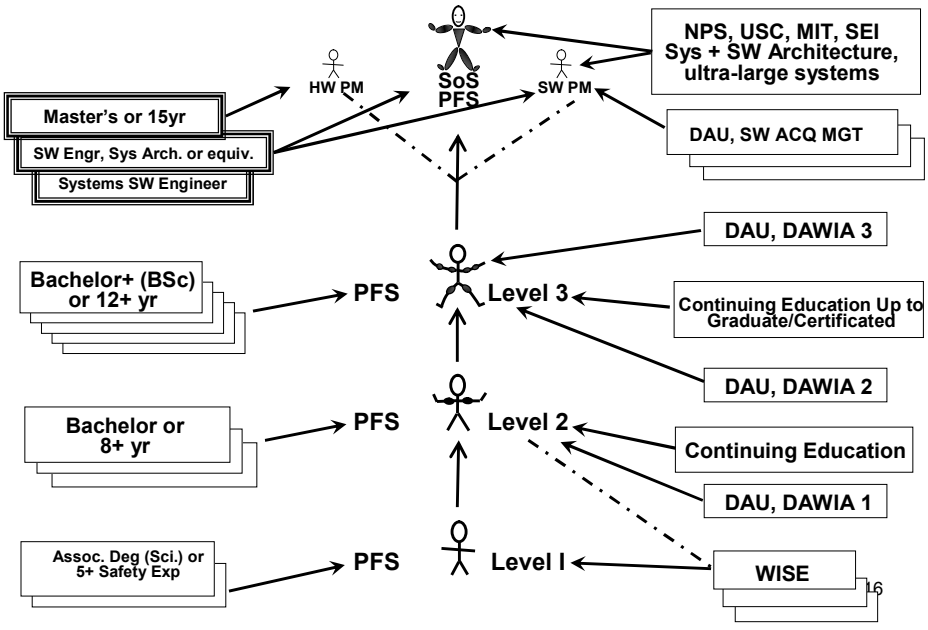

Systems Integration
Red lines/arrows are required but not yet determined
System Assurance (w/Anti-Tamper, etc.)
System Quality Assurance
System Configuration Management
Systems Safety Engineering
PM – cost, schedule, risk management
Systems Engineering
SEP
SoS/FoS Plan uidebook
SEP Planning Guide
Oct 2007?
To be included:
WSESRB, IA/CPI, IA/PPP
SETR
OSD SW Safety Guide
Sept2007
ASN(RDA), ASN(CHENG)
Info Assurance & Program Protection
Software Quality Assurance (SQA)
Software Configuration Management
Software Systems Safety Engineering
PM – cost, schedule, risk management
Software Engineering
SDP
OSD-DAU Module for SAM (PM level)
August2007
MIL-STD-882 & DOD 5000.2
OSD, NII
ASN (CHENG), OSD
WISE update (Engr Level)
Nov2007
USC Continuing Ed. (Engr Level)
May2007
ASN(CHENG)
OSD(AT&L)
OSD(AT&L)
MIL-STD-882
SSPP
Sequence metrics
Systems Safety Processes
HW
SW
metrics
NOSSA
SW Safety Handbook
14

Information Assurance



15

Principal-for-Safety (PFS) Certification Track



16

A System of Systems Interface Hazard Analysis Technique

FLTLT Patrick Redmond
Prof Bret Michael

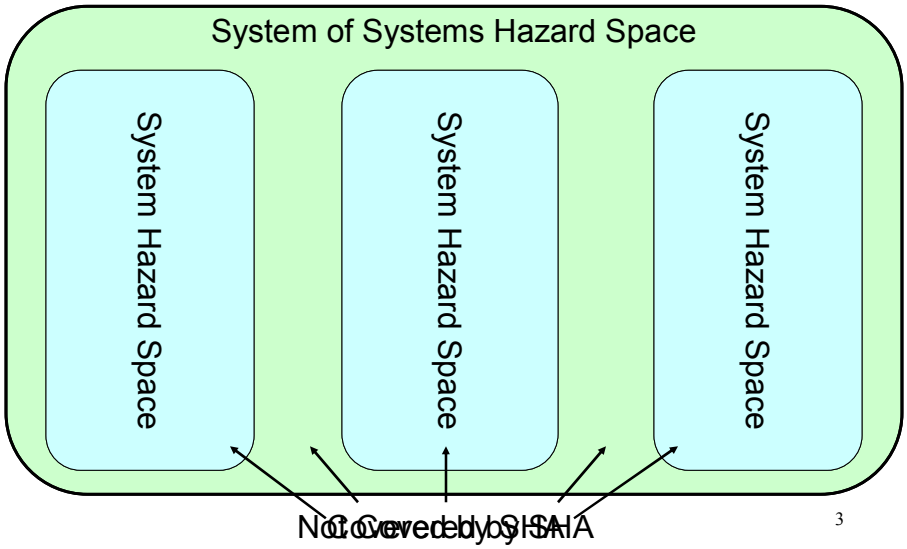
March 2007

Overview

- Systems of Systems Hazards
  - Definition
  - Types
- System of Systems Interface Hazard Analysis
  - System Architecture
  - System Model
  - Interface Analysis
  - Mishap Risk
- Concept Demonstrator

2

Systems of Systems Hazards



3

Systems of Systems Hazards

- Systems of Systems Mishaps:
  {SoS Mishaps} = Sum of {System Mishaps}

That is, the arrangement of systems into a SoS does not introduce any new mishaps.

4

Systems of Systems Hazards

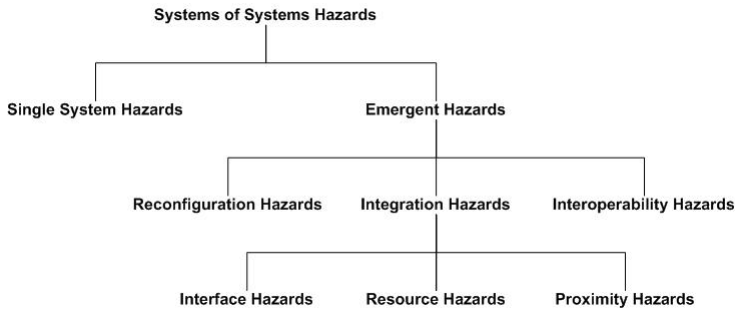
- Systems of Systems Hazards:
  {SoS Hazards} ≠ Sum of {System Hazards}

That is, the arrangement of systems into a SoS introduces new hazards.

Summary: The same mishaps exist, but there are new causes for them.

5

Systems of Systems Hazard Topology



6

Systems of Systems Hazards

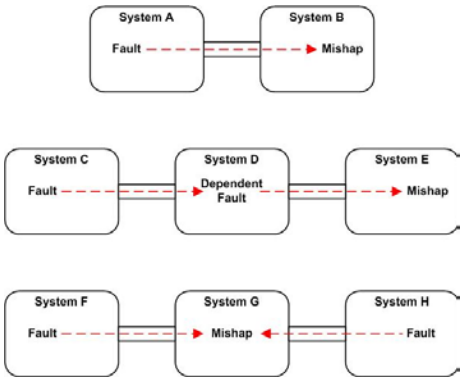
An ***emergent hazard*** is any hazard that may occur within a system of systems that is not attributable to a single system.

A ***single system hazard*** is any hazard that may occur within a system of systems that is attributable to a single system and may occur whether or not that system is operating within the system of systems context.

7

Systems of Systems Hazards

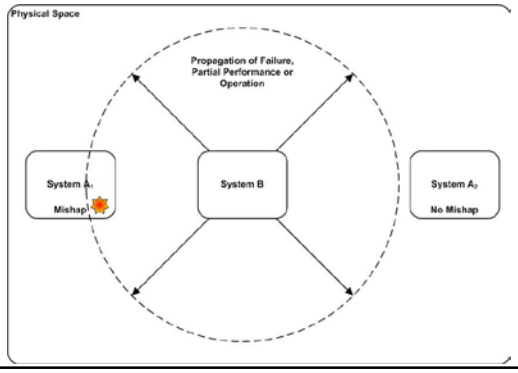
An ***interface hazard*** is a hazard in which one system causes a mishap in another system by transferring a failure or partial performance over a defined interface, possibly through another system.



8

Systems of Systems Hazards

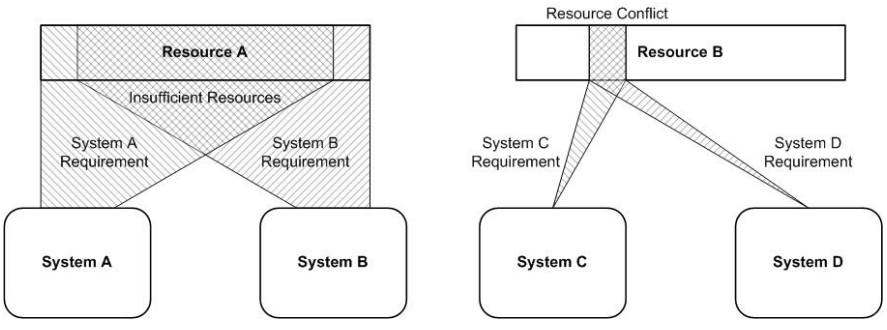
A ***proximity hazard*** is a hazard in one system that is caused by the operation, failure or partial performance of another system that is transferred to the victim system by a means other than a defined interface.



9

Systems of Systems Hazards

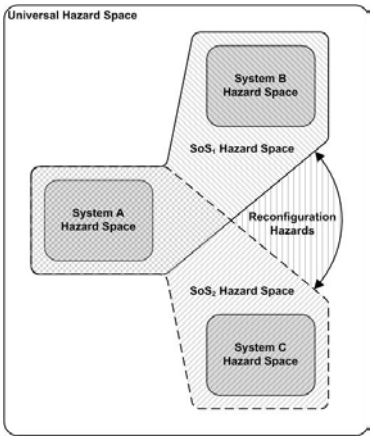
A ***resource hazard*** is a hazard that results from insufficient shared resources or resource conflicts



10

Systems of Systems Hazards

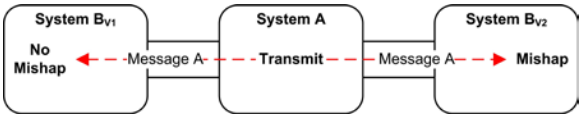
A ***reconfiguration hazard*** is a hazard that results from the transfer of a system of systems from one state to another.



11

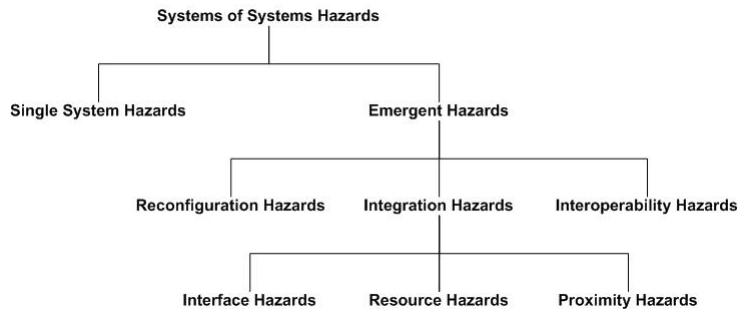
Systems of Systems Hazards

An ***interoperability hazard*** is a hazard that occurs when the command, response or data of one system is interpreted by a second system in a manner that is inconsistent with the intent of the first system.



12

Systems of Systems Hazard Topology



13

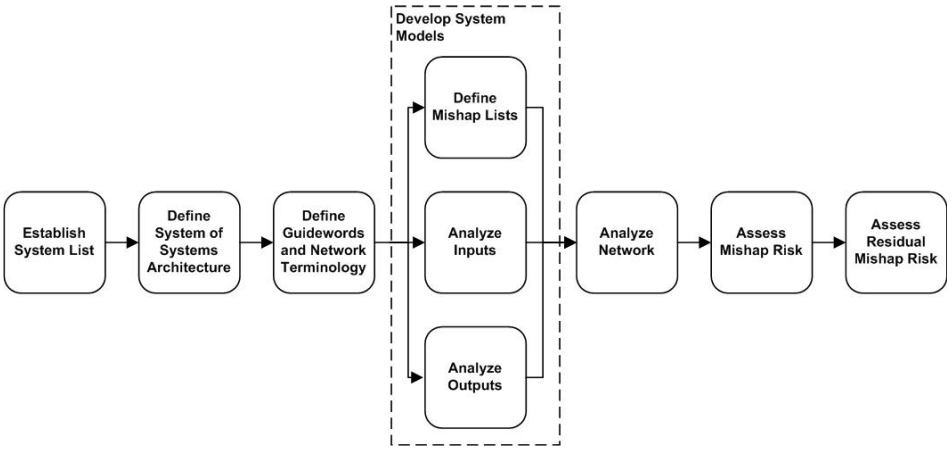
System Safety Process

- Documentation of the System Safety Process
- Identification of Hazards
- Assessment of Mishap Risk
- Identification of Mishap Risk Mitigation Measures
- Reduction of Mishap Risk to an Acceptable Level
- Verification of Mishap Risk Reduction
- Acceptance of Residual Mishap Risk
- Tracking of Hazards and Residual Mishap Risk

Systems of Systems cause problems for steps in red.

14

Interface Hazard Analysis Technique



15

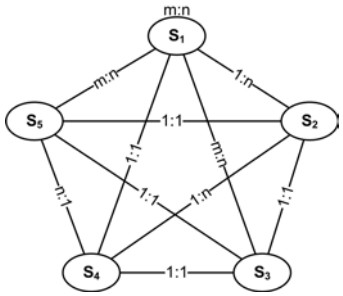
Establish System List

- List all systems that may be part of the SoS, including:
  - Systems that are always present,
  - Systems that may be present, and
  - Systems that are present sporadically.

Question: What systems may be present in the SoS?

16

SoS Architecture

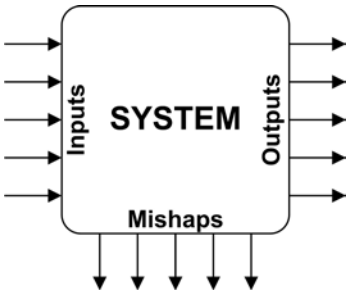


Question: How does System A interface with System B?

a) 1 to 1
b) 1 to n
c) n to 1
d) m to n

17

System Model Development



- Steps for Developing a System Model:
  - Identify System Mishaps
  - Identify System Input failures that can lead to a system mishap or a system output failure
  - Identify System Outputs that can fail as a result of an internal system failure

18

Mishap Identification

Question: What mishaps can occur within a given system?

Hint: System Mishaps should have been identified within individual system hazard analyses

19

Input Analysis

Question: What is the effect on the system if an input fails in a certain manner?

Use guidewords (corrupt, incomplete, absent, etc) to assess the impact of an input failure (i.e. does it cause a mishap? Does it cause an output to fail?)

Input + guideword = Input Failure > Mishap or Output

Are there combinations of failures that will lead to mishaps or output failures?

Do not consider whether another system could actually cause the input failure.

20

Output Analysis

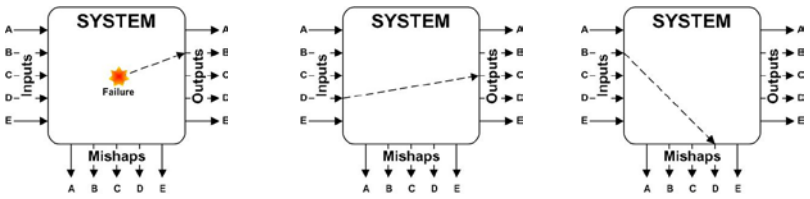
Question: What output failures can occur without the influence of another system?

Use guidewords to determine how the outputs can fail.

21

System Model Summary

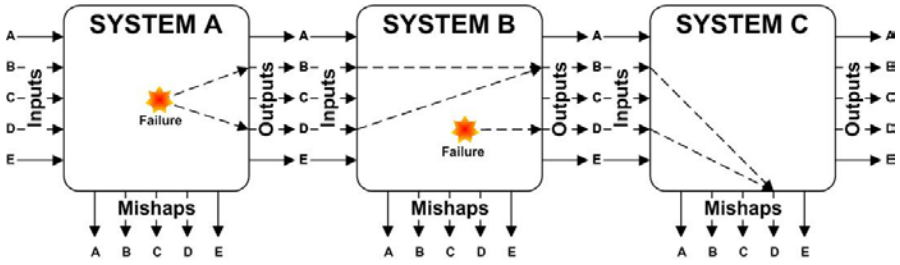
- A system model is:
  - A list of outputs that can fail and how they can fail
  - A list of links from specific failed inputs to a mishap
  - A list of links from specific failed inputs to a type of failed output



22

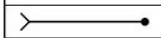
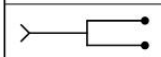
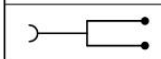
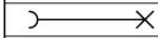
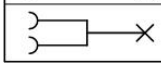
Finding Interface Hazards

Question: How can systems be combined to cause a mishap?



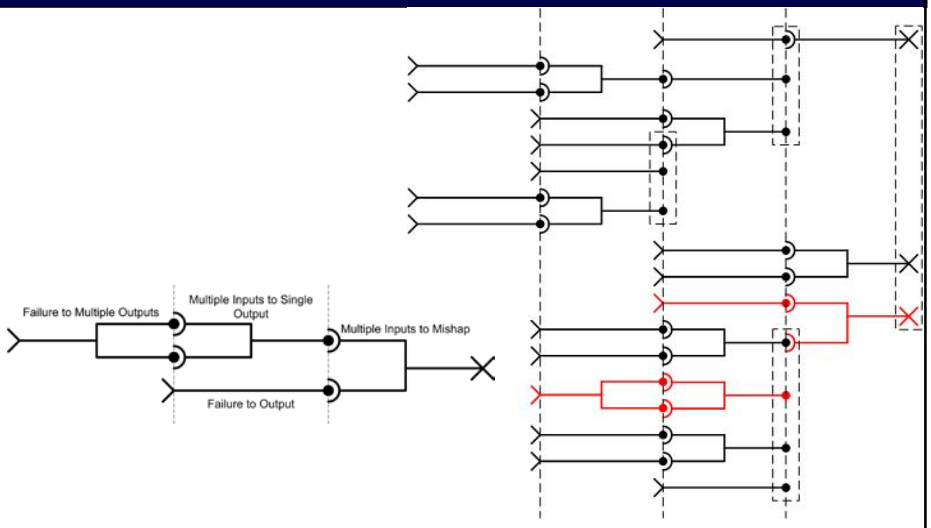
23

Interface Hazard Trees

Symbol	Meaning
	Failure
	Mishap
	Input
	Output
	Failure to Output Link
	Failure to Multiple Outputs Link
	Input to Output Link
	Multiple Inputs to Single Output Link
	Single Input to Multiple Outputs Link
	Input to Mishap Link
	Multiple Inputs to Mishap Link

24

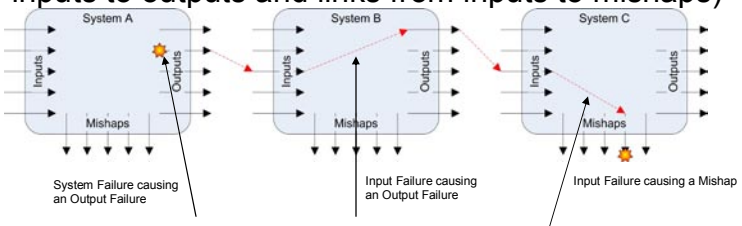
Interface Hazard Trees



25

Interface Hazard Priority

- Function of Probability and Consequence:
  - Consequence is the consequence of the mishap, which can be obtained from the relevant SHA
  - Probability is the combination of the probabilities of the hazard events (i.e. output failures, links from inputs to outputs and links from inputs to mishaps)



Each one has a probability of occurrence

26

Probability Combinations

Worst Case Scenario

	Frequent	Probable	Occasional	Remote	Improbable
Frequent	Frequent	Probable	Occasional	Remote	Improbable
Probable	Probable	Occasional	Remote	Remote	Improbable
Occasional	Occasional	Remote	Remote	Remote	Improbable
Remote	Remote	Remote	Remote	Improbable	Improbable
Improbable	Improbable	Improbable	Improbable	Improbable	Improbable

Best Case Scenario

	Frequent	Probable	Occasional	Remote	Improbable
Frequent	Occasional	Remote	Remote	Improbable	Improbable
Probable	Remote	Remote	Remote	Improbable	Improbable
Occasional	Remote	Remote	Improbable	Improbable	Improbable
Remote	Improbable	Improbable	Improbable	Improbable	Improbable
Improbable	Improbable	Improbable	Improbable	Improbable	Improbable

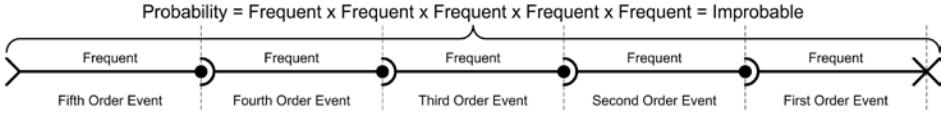
Average

	Frequent	Probable	Occasional	Remote	Improbable
Frequent	Probable	Occasional	Remote	Remote	Improbable
Probable	Occasional	Remote	Remote	Improbable	Improbable
Occasional	Remote	Remote	Remote	Improbable	Improbable
Remote	Remote	Improbable	Improbable	Improbable	Improbable
Improbable	Improbable	Improbable	Improbable	Improbable	Improbable

27

Residual Risk

- SoS are complex, how to assess how much risk remains?
- Function of:
  - Known Hazards
  - Hazards within system models but not identified by search
  - Hazards left out of system models



28

Residual Risk - Example

- For example, using the average probability combination:
  - Highest Probability with three elements:
    - Frequent x Frequent x Frequent = Occasional
  - Highest Probability with four elements:
    - Frequent x Frequent x Frequent x Frequent = Remote
- If residual risk needs to have all hazard probabilities at remote or below, then all combinations of three elements must be assessed.
- The maximum allowable hazard probability will depend upon the acceptable priority threshold, the method for combining probabilities and consequences, and the highest consequence.

29

Accommodate New Systems

- SoS will have new systems added over time
- The interface hazard analysis technique must easily accommodate this
- Process above can do this because:
  - It is automated, hazards can be generated quickly once data is known. Combinations are not explored by hand.
  - System models are independent of the surrounding systems (i.e. failed inputs are linked to mishaps whether or not that failure can occur, it might not be possible currently, but may be in the future).

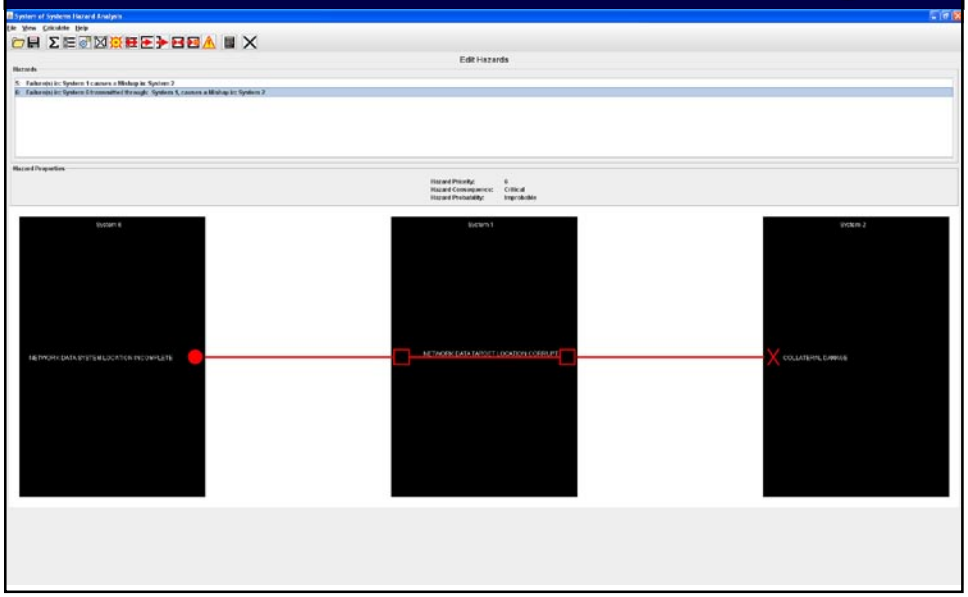
30

Concept Demonstrator

- Allows the user to enter system model data
- Searches the model data for SoS hazards
- Is not meant to be used, just to demonstrate that the process can be easily implemented.
- Not complete:
  - Network analysis algorithm is simplified
  - Does not accommodate common mode failures
- So far:
  - Successfully identifies some hazards

31

Questions?



Safety and Security in Secure Software Engineering

Samuel T. Redwine, Jr.
James Madison University
701 Carrier Drive
Harrisonburg, VA 22807

Abstract- In recent years, the software safety community has more examples of successful experience with producing high-confidence software than does the software security community. The safety community's experience provides lessons for software security practitioners, but as usually approached the engineering safety problem differs from the security one in a critical way – it presumes non-existence of maliciousness and thereby the appropriateness of probabilistic analysis. The existence of maliciousness and security weaknesses could lead to the similar adverse consequences as those usually addressed in safety engineering meaning "safety" cannot be provided unless security is addressed.

I. SAFETY AND SECURITY

Today, security is a concern for most systems as software has become central to the functioning of organizations and physical systems with much of it is directly or indirectly exposed to the Internet or to insider attack as well as to subversion during development, deployment, and updating. Though safety-oriented systems so exposed now must also face the security problem, often traditional computing safety engineering does not address maliciousness and its perverse effect on probabilistic analyses.[1] [9]

This engineering analysis technique issue may be operationally more significant than whether something is labeled a safety or a security concern as, in practice, what is labeled what (including neither) is not strictly a function of maliciousness, type or size of adverse consequence, illegitimacy, or even the existence of a criminal penalty.

[1] However, physical and operational safety have recognized closely related security concerns, for example the time needed to break into a nuclear plant.

II. PROBABILITY VERSUS POSSIBILITY

The patterns of occurrences of "natural" events relevant to safety are described probabilistically for engineering purposes. The probability of a natural event contrasts with the need for concern for the possibility of an intelligent, malicious action. For example, a standard tactic in conflicts is to attack where and when the opponent least expects it – that is where the defender has assigned the lowest probability or risk. This distinction is central to the difference between facing safety hazards versus security threats (threatening entities, conditions, events, and consequences). [9]

III. COMBINING SAFETY AND SECURITY

One can write requirements for safety and security that look quite similar in form.[1] Here is a highly abstract and condensed example that shows elements of this commonality.

- The system shall have
  - limited adverse consequences
  - limited related uncertainty
- The combined effect of these shall provide basis for an engineering conclusion that system will meet, is adequately progressing toward meeting, or has met its requirements or claims regarding risk and consequences criteria and objectives
- These claims and the valid arguments with supporting evidence, which justify them, shall be documented and reviewed in a well-organized, defensible, and auditable "Assurance Case."
- This Assurance Case shall provide adequate grounds for stakeholders' acceptable confidence and rationally justified decisions
- The assurance case shall
  - Identify system's environments and conditions throughout its lifespan; and capabilities, acts, events, and conditions

within the system and its environments that may threaten or damage the system or the interests of relevant stakeholders.
  - Be planned, developed, and managed concurrently and integrally with the system, and maintained throughout the lifespan of the system.
  - Make verifiable claims supported by thorough, robust arguments and valid evidence of adequate quality and known relevance and meaningfulness and include any contraindicating arguments and evidence.
  - Be planned, developed, and maintained using suitable, integrated system/software processes, resources and means, environment, management, and time of quantity and quality shown to be sufficient for achieving claims and ensure validity

When both safety and security are required, a number of areas are candidates for partially combining safety and security engineering concerns including:

- Goals or claims including sub-goals or sub-claims,
- Constraints on system behavior,
- Principles,
- Solutions,
- Activities and processes,
- Assurance case:
  - Goals/claims,
  - Assurance arguments,
  - Evidence,
- Correctness
- Evaluations and certifications.

Many people have pointed out commonalities. [2] [7] [8] The SafSec effort provides guidance on one way to do this at least for assurance cases [5] [6].

In recent years, the software safety community has been more advanced in its thinking, for example [3] [4], and has more examples of successful experience with producing high-confidence software than does the software security community. The safety community's experience provides lessons for software security practitioners, but the traditional engineering safety problem differs from the security one in a critical way – it presumes non-existence of maliciousness.

IV. REFERENCE

[1] Firesmith, D. "Common Concepts Underlying Safety, Security, and Survivability Engineering." *Technical Note CMU/SEI-2003-TN-033*. 2003

[2] Lautieri, S., Cooper, D., and Jackson, D. "SafSec: Commonalities Between Safety and Security Assurance." *Proceedings of the Thirteenth Safety Critical Systems Symposium* - Southampton, 2005.

[3] Leveson, Nancy. "A Systems-Theoretic Approach to Safety in Software-Intensive Systems," IEEE Transactions on Dependable and Secure Computing 1, 1 (January-March 2004): 66-86, 2004.

[4] Ministry of Defence. Interim Defence Standard 00-56, Safety Management Requirements for Defence Systems Part 1: Requirements, 17 December 2004.

[5] "SafSec Methodology: Guidance Material", *SafSec: Integration of Safety and Security.* Available at: http://www.praxis-his.com/safsec/safSecStandards.asp.

[6] "SafSec Methodology: Standard:." *SafSec: Integration of Safety and Security.* Available at: http://www.praxis-his.com/safsec/safSecStandards.asp.

[7] Stavridou, V. and Dutertre, B. "From Security to Safety and Back." *Computer Security, Dependability and Assurance: From Needs to Soultions, 1998, Proceeding.* 1998

[8] Stoneburner, Gary. "Toward a Unified Security/Safety Model." *Computer.* Volume 39, Issue 8. Pages 96-97. August 2006

[9] Redwine, S. (Editor). *Secure Software Assurance: A Guide to the Common Body of Knowledge to Produce, Acquire, and Sustain Secure*

Software Version 1.1 US Department of Homeland Security, September 2006.

V. ABOUT THE AUTHOR

Samuel T. Redwine, Jr.
James Madison University
redwinst@cs.jmu.edu
540-568-6305

Samuel Redwine has had an extensive career of practice, scholarship, and research in the fields of software engineering and more recently software security. With a history of numerous publications and presentations, he has serviced as General Chair of the International Symposium on Secure Software Engineering and workshops as well as on numerous program committees. He is involved in a number of related efforts including ones associated with the Object Management Group, National Defense Industry Association, National Security Agency, and the Department of Homeland Security.
In addition, he has helped spread education in secure software including helping found JMU's Secure Software Engineering masters program.

Safety and Security

Samuel T. Redwine, Jr.
James Madison University
IWSS07
March 15, 2007

Topics

- Some Commonalities regarding Danger
- Traditional Security and Safety
- Using Probability?
- Combining Safety and Security
  - Shared Abstract Requirement
  - Shared Concerns

Some Commonalities regarding Danger

Concern for Danger and Damage make Safety and Security Bedfellows

Desirable Characteristics of a Software or System Solution

- Better
  - Functionality
  - Performance
    - Capacity, throughput, and speed
  - Efficiency – Benefits and Costs
  - (Less) Danger
  - Opportunity
  - Pleasingness
  - Certainty
- Compliance
  - Contracts, laws and regulations, and policy

Duality of Events and Conditions

- **Bad Events** to eliminate, limit, reduce, or manage ⟺ **Good conditions** to be established and preserved
  - Bad "Real-World" Events ⟺ "Real-World" conditions
  - Bad System-Environment Interactions ⟺ Flow constraints
  - Bad Computing Asset-related Events ⟺ Asset properties (e.g. Confidentiality, Integrity, Availability)
  - Bad System Actions ⟺ System states that are **not preconditions** for act
  - Actions needed to prevent, avoid, … respond to bad actions and events ⟺ Ability to establish and maintain capability

Examples

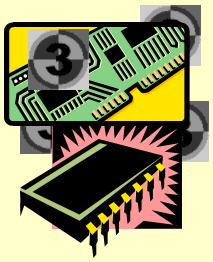
	Bad Event	**Good Condition Always True**
Real World	Reactor Meltdown	Reactor Temperature within limits
System-Environment Interactions	Dangerous human instruction	All human operators properly trained All commands allowed to go to reactor are predicted to be benign
Computing Assets	Table of limits or History data corrupted	Table of limits changed only by authorized entities and history data never changes
Software System	Operation involving outside or asset is not logged	Every operation involving an asset has proper authorization and is logged
Enabling Functionality	Identification management lacking	Required functionality is available, correct, and tamper proof (and also needs to be not bypassable)

Requirements Frames+

Computing System-
Environment Interaction

Consequences

Computing System



Software and Hardware

Engineering Representation

Specifications

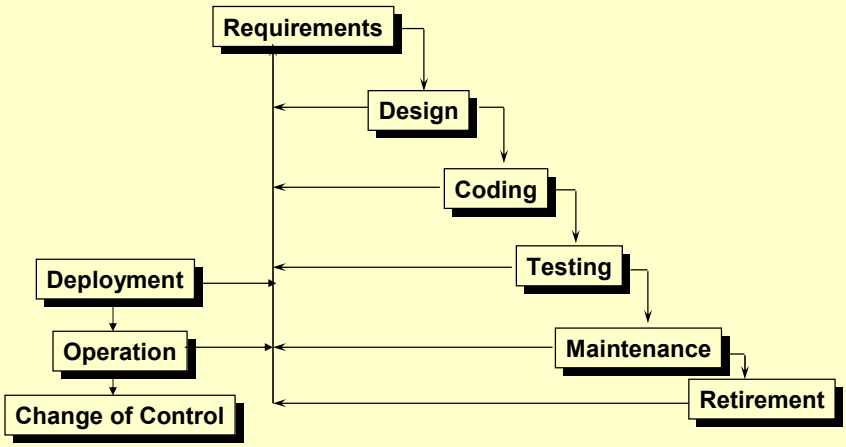
- Behavior (e.g. Functionality)
- Constraints on behavior of functionality

Security properties;
Safety properties;
emergent system properties

For example, constraints in forms of:

- Allow only authorized authorities to order reactor startup
- Never allow anyone but good guys to change sensitive data
- Security functionality is not bypassable

Attacks, Mistakes, and Failures Possible in All Activities and Environments



Summary

- Safety and security share much in common
  - Ultimately about adverse consequences and their uncertainty
  - Duality of avoiding bad events and preserving good conditions
    - Bad events cause, allow, facilitate, or contribute to adverse consequence
    - Avoid preconditions for bad events
  - System in danger all its life

Traditional Security and Safety

- Security
  - Adverse consequences
  - Non-maliciousness
  - Illegitimacy
  - Maliciousness
- Safety
  - Adverse consequences
  - Non-maliciousness

Behavior can be result of outside entity or inherent behavior of system or software.

Security Properties – CIA+

- Confidentiality: preventing unauthorized disclosure
- Integrity: preventing unauthorized alteration
- Availability: preventing unauthorized destruction or denial of access or service
- Accountability: knowing what entity did what when
  - Non-repudiation: ensuring the inability to deny the ownership of prior actions

Identification: known identities needed for entities as much of security is about who can do what when
Authentication: verifying identity ensuring entity identified correctly

Maliciousness

- Existence of maliciousness does not make non-malicious problems go away
- Performing to specification
  - Not probabilistic reliability and availability
  - Adversaries often attack where least expected
  - Anything could happen
- Adversary wants best thing for her/himself – may include wanting to make the worst possible thing happen to you at the worst time

Using Probability

- Can malicious behavior be validly modeled probabilistically?
  - Reasonably frequent similar malicious behavior generally can be modeled probabilistically. E.g. car theft, household burglaries, script kiddies
  - However, serious adversaries tend to, **"Attack where, when, and how their opponent least expects"**
    - That is, where probability is judged to be low

Result of Murphy's law happens not evidently but deliberately ☹

Kinds of Reasoning Systems

- "Quantitative"
  - Deterministic
    - E.g. formal proofs
  - Non-deterministic formal systems for reasoning
    - Probabilistic
    - Game theoretic
      - E.g. minimax
    - Other uncertainty-based formal systems of reasoning
      - E.g. worst case, fuzzy sets, others used in AI
- Qualitative
  - E.g. staff skill and experience, compliance with standard, qualitative statements of event causality

Summary

- Security (of bits) includes
  - Confidentiality,
  - Integrity,
  - Availability, and
  - Accountability,
  - But ultimately it is about adverse consequences and their uncertainty
- A common way to think about security-related software behavior is with constraints
- Probability-based analysis may not work for maliciousness

Shared Abstract Requirement - 1

- The system shall have
  - limited adverse consequences
  - limited related uncertainty
- Together these shall provide a basis for an engineering conclusion that system *meets* its requirements or claims regarding criteria and objectives related to risk and consequences
  - *meets* = will meet, is adequately progressing toward meeting, or has met – depending on time

Shared Abstract Requirement - 2

- These claims and the valid arguments with supporting evidence, which justify them, shall be documented and reviewed in a well-organized, defensible, and auditable "Assurance Case."
- This Assurance Case shall provide adequate grounds for stakeholders' acceptable confidence and rationally justified decisions
- System (including assurance case) shall be planned, developed, and maintained in a manner sufficient for achieving claims and ensuring adequacy and validity of assurance case
  - using suitable, integrated system/software processes, resources and means, environment, management, and time shown to be sufficient

Shared Abstract Requirement

- The assurance case shall
  - Identify what may threaten or damage the system or the interests of relevant stakeholders
    - within system's environments and conditions throughout its lifespan; and for all capabilities, acts, events, and modes
  - Be planned, developed, and managed concurrently and integrally with the system, and maintained throughout the lifespan of the system.
  - Make verifiable claims supported by
    - thorough, robust arguments
    - valid evidence of adequate quality and known relevance and meaningfulness
      - include any contraindicating arguments and evidence, and
      - having limited assumptions (non-critical, weak)

Candidates Concerns for Partially Combining Safety and Security

- Goals or claims including sub-goals or sub-claims
- Constraints on system behavior
- Principles
- Solutions
  - E.g. Fault tolerance
- Activities and processes
- Assurance case
  - Goals/claims
  - Assurance arguments
  - Evidence
  - Assumptions
- Predictability
- Correctness
- Evaluations and certifications

In Closing

- Abstractly and in practice safety and security have similarities
- Maliciousness is hard to totally discount anywhere and can make probabilities unknowable/unsuitable
- An important question in engineering practice is when a probabilistic approach is appropriate

Questions and Discussion

Analogize	*Conceptualize*	*Hypothesize*	*Ponder*
Analyze	*Conjecture*	*Infer*	*Propose*
Apply	*Discover*	*Imagine*	*Question*
Agree	*Discriminate*	*Integrate*	*Reason*
Argue	*Estimate*	*Invent*	*Recount*
Assert	*Evidence*	*Judge*	*Specialize*
Calculate	*Examine*	*Link*	*Solve*
Caution	*Explain*	*Measure*	*Suppose*
Claim	*Extrapolate*	*Observe*	*Theorize*
Clarify	*Foresee*	*Opine*	*Validate*
Conceive	*Generalize*	*Organize*	*Verify*

Some Terms for Bad Things

- **Specification fault**
- Violations of specification
  - Violation coming from outside
    - **Mistake, accident, mishap, act of nature, subversion, penetration, or attack**
  - Violation in static representation of system
    - **Fault, vulnerability** (or more casually "defect")
  - Violation of constraints on dynamic system state
    - **Error**
  - Violation from inside crossing system boundary – visible outside
    - **Failure**
- Result – inside or outside system
  - **Adverse consequence, cost, loss**
  - **Disclosure, corruption, denial, repudiation, compromise**
- Preconditions for violation or adverse consequence
  - **Improper authorization, hazard, contributing factor, bypassable, subvertable** (Unsecured, unsafe)
- Propensity toward Bad Things
  - **Weakness, bad practice, error prone, failure prone**

Competency Software Safety Requirements for Navy Engineers
14 March 2007

Brian Scannell
BS Engineering Science, University of Louisville, 1990
MENG Computer Engineering, University of Louisville, 1995
MBA, University of Louisville, 2004

Paul Dailey
BS Electrical and Computer Engineering, University of Louisville, 2004
MS Systems Engineering, Naval Postgraduate School 2006

ABSTRACT

The Navy currently has no formal certification for Safety Engineers concentrating in software safety. NOSSA has led an effort to educate personnel regarding the development and support of Naval Weapon Systems. The WISE training tool is a step in the right direction, but further formal training is needed to support experience in software safety. There is a need to evade the case of untrained software safety engineers that are arbitrarily appointed tasks. This should not be based on education or experience alone but rather a combination of experience, education, and certification. A documented certification process will only improve systems required to be safe that depend on software. This document provides several options to obtain a solid software and systems safety background for Navy applications.

Naval Ordnance Safety and Security Activity

The Naval Ordnance Safety and Security Activity (NOSSA) is a field activity of the Naval Sea Systems Command (NAVSEA). NOSSA manages all aspects of the Department of the Navy (DoN) Explosives Safety Program. As the NAVSEA technical authority for Explosives Safety, NOSSA is responsible for providing technical policies, procedures and design criteria associated with weapons systems safety, including software safety across the warfare disciplines. NOSSA manages all programmatic policy requirements for the five major DoN Explosives Safety Program component programs; Ordnance Safety and Security, Weapons and Combat System Safety, Ordnance Environmental Support Office, Insensitive Munitions Office, and Weapons and Ordnance Quality Evaluation.[1]

NOSSA Certification and Training

The NOSSA objective for certification and training is to establish a reasonable and recognizable assurance of the system safety competencies necessary in managing today's complex systems and Research, Development, Test & Evaluation (RDT&E) efforts. The ultimate goals of NOSSA's certification process are to promote those qualitative characteristics required of Naval and support contractor personnel engaged in system safety practices, and to enhance the system safety engineering processes within existing and future Navy acquisition programs. In complying with current requirements, it is critical that a process is established by which personnel qualifications and training can be measured and confirmed.

WISE Online Training

The Weapon System Explosives Safety Review Board (WSESRB) Interactive Safety Environment (WISE) training program provides the medium for achieving Principal for Safety (PFS) certification. Through a series of testable modules, a potential candidate can gain access to the body of knowledge required to perform as an effective PFS for DoN Programs.[2]

---

[1] http://www.nossa.navsea.navy.mil/

[2] NOSSA secure website

Software Safety Handbook

In 1999, the Joint Software System Safety (SSS) Committee developed a handbook to provide management and engineering guidelines to achieve a reasonable level of assurance that the software will execute within the system context with an acceptable level of safety risk. The handbook is both a reference document and management tool for aiding managers and engineers at all levels, in any government or industrial organization. It demonstrates "how to" in the development and implementation of an effective SSS process. This process minimizes the likelihood or severity of system hazards caused by poorly specified, designed, developed, or operation of software in safety-critical applications.[3]

GRADUATE LEVEL PROGRAMS

There are a number of graduate level courses and programs offered in system safety. These include the University of Southern California (USC), Embry Riddle University, the University of York, Texas A&M, and Massachusetts Institute of Technology (MIT). MIT has a program that concentrates on software engineering (SERL, or Software Engineering Research Laboratories).

University of Southern California

The USC Viterbi School of Engineering provides a four day course in software safety in addition to a two week course in systems engineering. These courses support certification in USC Aviation Safety and Security.[4]

The Software Safety course presents philosophies and methods of developing and analyzing software and highlights managing a software safety program. Software design principles are taught to create programs that are fault tolerant and acceptably safe. Several software hazard analyses methods are evaluated, including Fault Tree/Soft Tree, Software Sneak Analysis and Petri Nets. The course objective is to provide an understanding of the nature of software hazards, root causes, and the methods by which these hazards may be prevented or discovered. The course also provides instruction using administrative methods and documentation needed to establish and manage a software safety program. Providing evidence for a safety case or proof is also covered. This course is designed for systems managers and engineers, systems safety engineers and software engineers who are involved with developing systems that possess major software components and are responsible for the safety. Recommendations for preparation for this course include attending the System Safety Engineering course and some understanding of software.[5]

Embry Riddle University

Embry Riddle University doesn't offer a specific software safety course, but it does have a Bachelor of Science and Master of Science degree in Safety Science. The Bachelor of Science degree is taught at the Daytona Beach campus.[6] The Master of Science degree is taught at the Prescott, AZ campus.[7] Both the Bachelor and Master degree concentrate on the aeronautical field.

The University of York

---

[3] Software System Safety Handbook, A Technical & Managerial Team Approach, Joint Services Computer Resources Management Group, U.S. Navy, U.S. Army, and the U.S. Air Force

[4] http://viterbi.usc.edu/aviation/aviation_cert_program.htm

[5] http://viterbi.usc.edu/aviation/sft.htm

[6] http://www.erau.edu/db/degrees/b-safetyscience.html

[7] http://www.erau.edu/omni/pr/academicorgs/prssd/

The University of York provides a Systems Safety Engineering (SSE) Certificate two year course. The Two year course has six modules. Each module is taught full time in York for one week. Its associated assessed exercise, which may be completed on or off site, takes approximately 35 hours in addition. All assessed exercises are open, comprising a report, case study, or documented piece of software.[8]

NPS Monterey – Weapons System Software Safety

Naval Postgraduate School in Monterey offers a weapons system software safety course, a requirement for a master's degree in Systems Engineering. SW4582[9] provides the foundation for Software Systems Safety. The course focuses heavily on the Software Engineering aspects of the discipline; the content injects enough Systems Safety Engineering principles to ensure that the graduates fully understand their responsibility in the overall system development process.

University of Washington

The University of Washington College of Engineering periodically offers software systems safety courses, in addition to system safety management and reliability analysis. The software systems safety course is a five day course that provides the knowledge needed to implement a practical software safety effort for maximum impact on design and test activities. [10]

SYMPOSIUMS AND CONFERENCES

The International System Safety Conference (ISSC) is held annually during the summer. The Joint Weapon System Safety Conference (JWSSC) is held in conjunction with the ISSC. The 2007 ISSC/JWSSC is scheduled August 13-17 in Baltimore, MD.[11]

The newly created Technical Committee on System Safety under the IEEE System Society is holding a series of annual international workshops on issues relating to safety of systems of national and global significance. The first event is being held in March 2007 at the Naval Postgraduate School.[12]

The IEEE International Symposium on Dependable Autonomic and Secure Computing (DASC) is held annually. The 2007 Symposium is scheduled for Sept 25-27, 2007 at Loyola College Graduate Center, Columbia, MD.

The International Conference on Computer Safety, Reliability, and Security, or SAFECOMP, is an annual event covering the state-of-the-art, experience and new trends in the areas of computer safety, reliability and security regarding dependable application of computer systems. SAFECOMP provides ample opportunity to exchange insights and experience on emerging methods and practical application across the borders of different disciplines. The 2007 SAFECOM is scheduled for Sept 18-21, 2007 in Nuremburg, Germany.[13]

Safeware© System Safety for Software-Intensive Systems

---

[8] http://www.cs.york.ac.uk/gsp/sse_cert.php

[9] http://www.nps.edu/DL/NPSO/courses/course_descriptions.html#SW4582

[10] http://www.engr.washington.edu/epp/safety/sss.html

[11] http://www.system-safety.org/~am_2007/

[12] http://www.ieeesystemscouncil.org/conferences.html#iwss

[13] http://www.safecomp.org/

Safeware Corporation offers a one week class covering fundamental concepts and techniques in building and ensuring safety, with particular emphasis on those aspects of complex systems not handled well by traditional system safety approaches, such as software and human-computer interaction. [14]

DAU Online Training

Defense Acquisition University (DAU) offers the Systems Safety for Systems Engineers course (CLE009). This is an online 3.5 hour course module which shows how the MIL-STD-882D methodology is integrated into the Department of Defense systems engineering process for eliminating environment, safety, and occupational health hazards or minimizing the associated risk. It uses the systems engineering V-model to identify the key system safety activities that are conducted during each phase of the system's life cycle.

CERTIFICATION

Several options for safety certification are available. These include Certified Safety Professional Certification, PFS certification by NOSSA, and Certified Functional Safety Expert (CFSE) certification.

Certified Safety Professional

The Board of Certified Safety Professionals (BCSP) provides certification in safety which combines education, experience and a certified exam.[15]

PFS Certification and Designation by NOSSA

A PFS is recognized as certified to perform associated system safety managerial duties only upon successful completion of the certification process outlined in NOSSAINST 12410.5. A PFS certification provides a quantified and recognizable assurance of system safety competencies to program management. While PFS Certification is a professional mark of distinction, it is not, and shall not be used as, an employment designation.

Certification as a PFS should not be confused with designation as a PFS. The designation as an Acquisition Programs PFS is a formally delegated authority, as required by OPNAVINST 5100.24A, given in writing by the acquisition program's management. The Acquisition Program PFS is both the technical authority regarding matters of system safety, and the professional conduit into the system safety body of knowledge, the system safety community, and the acquisitions lifecycle.

NOSSA has defined a certification path for PFS at three levels. The three PFS Certification categories available are PFS High, Medium, and Low Certification.

Certified Functional Safety Expert Governance Board

The CFSE governance board was formed to improve the skills and formally establish the competency of those engaged in the practice of safety system application in process and manufacturing industries. The CFSE board offers exams which provide two levels of certification, a CFSE and CFSP. Prospective members can select a specialization, which includes Process Industry Safety, Machinery Safety, Safety Hardware Development, and Safety Software Development. In order to qualify to take the tests, candidates must have at least ten years experience in safety, but are given years credit for level of education. [16]

---

[14] http://www.safeware-eng.com/services/training.htm

[15] http://www.bscp.com

[16] http://www.cfse.org

CONCLUSIONS/RECOMMENDATIONS

This position paper was written to identify the options available to obtain system safety training, specifically in the Software Safety field. Formalized training is important, which provides a good foundation for software safety. In addition to training, certified software safety engineers should also have a mentor assigned, attend a safety board presentations to gain experience, and also provide a report of training experiences. Listed are recommendations to create better software safety engineers:

1. The WISE training tool must be required for software safety engineers. This system would document the Navy specific training.
2. The DAU system safety modules may be duplicate information, but software safety engineers should also be required to take all available safety courses offered. These applications are offered at the Department of Defense level. It is the hope that the DAU online and site coursework will be expanded in the safety field.
3. At least six hours of accredited course work from a post graduate program should be required.
4. A mentor assignment should be required for software safety engineers. It is important to obtain a mentor within the systems safety community to share experiences, knowledge and wisdom about safety, and to guide and assist less experienced safety personnel. The mentor can also provide training opportunities, in addition to sharing experiences that would specifically benefit a person desiring to broaden their knowledge in systems and software safety. A mentor/student relationship is instrumental in developing effective safety engineers.
5. Periodic attendance to safety related symposiums and conferences should be required. This gives the software safety engineer the opportunity to keep current with safety issues and solutions available.
6. Participation on SSSTRP and WSESRB should be required. Before software certification is granted, it is invaluable experience to participate or attend as a guest of the SSSTRP and WSESRB. The participation should include a related and non related system, to give the software safety engineer broader experience to systems outside his experience.
7. A safety engineer should be required to perform, or assist in preparing a software safety analysis to gain hands on experience.
8. The safety engineer should be required to provide an informal exit presentation or report of his training experiences. This will be valuable feedback to provide better future training experiences.

Competency Software Safety Requirements for Navy Engineers

Brian Scannell / Paul Dailey

NSWC PHD Louisville

March 2007

Overview

- US Navy currently has no formal engineering Software Safety certification process
- Software Safety engineers are often appointed based on either availability, experience, or education alone
- Certifications, training tools, courses, and workshops are available from various sources
- A documented quantitative certification process for DoN engineers will only improve safety for software intensive systems

NOSSA Background

- Naval Ordinance Safety and Security Activity is a field activity for NAVSEA
- NOSSA manages all aspects of the Navy's Explosives Safety Program
  - Ordinance Safety and Security
  - Weapons and Combat System Safety
  - Ordinance Environmental Support Office
  - Insensitive Munitions Office
  - Weapons and Ordinance Quality Evaluation

NOSSA Certification

- Promotes qualitative characteristics required by Safety personnel
- Enhances System Safety within the Systems Engineering Process for Navy acquisition programs
- In complying with these requirements, it is critical that a quantitative process is established

PFS and PFS Certification

- Principal for Safety (PFS) Certification is obtained via the Weapons System Explosives Safety Review Board (WSESRB) Interactive Safety Environment (WISE) online training and is not used as an employment designation
- A PFS is appointed for each DoN acquisition program by program management

Other Certifications

- The Board of Certified Safety Professionals (BCSP) provides certification combining education experience and examinations
- The Certified Functional Safety Expert Governance Board provides certification based on tests
  - Minimum of 10 years of experience in safety is required to take certification exam but years of credit can be given for level of education

Graduate Level Programs

- University of Southern California – 4 day course in Software Safety
- Naval Postgraduate School – 3 credit hour Weapon System Software Safety course
- Massachusetts Institute of Technology – Software Engineering Research Laboratories program
- Texas A&M – Systems Safety Program

Symposiums & Conferences

- International System Safety Conference (ISSC) / Joint Weapons Systems Safety Conference (JWSSC) are held annually
  - 07 session is August 13-17 in Baltimore, MD
- IEEE International Workshop on System Safety
- IEEE International Symposium on Dependable Automatic and Secure Computing (DASC) is held annually
  - 07 session is September 25-27 in Columbia, MD

Other Training

- Safeware® System Safety for Software Intensive Systems – 1 week class
- Defense Acquisition University (DAU) online training for Systems Safety for Systems Engineering

Recommendations for DoN Safety Engineers

- Require the WISE training tool for Navy specific training
- DAU system safety modules should be taken along with any other available safety courses
- Minimum of 6 hours of accredited coursework for Software Safety
- Periodic attendance to Safety related symposiums and conferences

Recommendations for DoN Safety Engineers (cont)

- A mentor assignment should be required for new safety engineers to share experience, provide guidance and promote training opportunities
- Participation on Software System Safety Technical Review Panel (SSSTRP) and WSESRB should be required
- Participation in a software safety analysis for hands-on experience
- Engineer should provide an informal exit presentation or report of training experiences to provide feedback needed to continuously improve process

Biologically-Inspired Concepts for Autonomic Self-Protection in Multiagent Systems

Roy Sterritt and Mike Hinchey

University of Ulster
School of Computing and Mathematics,
Jordanstown Campus, BT37 0QB
Northern Ireland
r.sterritt@ulster.ac.uk

NASA Goddard Space Flight Center
Software Engineering Laboratory
Greenbelt, MD 20771
USA
michael.g.hinchey@nasa.gov

Abstract. Biologically-inspired autonomous and autonomic systems (AAS) are essentially concerned with creating self-directed and self-managing systems based on metaphors from nature and the human body, such as the autonomic nervous system. Agent technologies have been identified as a key enabler for engineering autonomy and autonomicity in systems, both in terms of retrofitting into legacy systems and in designing new systems. Handing over responsibility to systems themselves raises concerns for humans with regard to safety and security. This paper reports on the continued investigation into a strand of research on how to engineer self-protection mechanisms into systems to assist in encouraging confidence regarding security when utilizing autonomy and autonomicity. This includes utilizing the apoptosis and quiescence metaphors to potentially provide a self-destruct or self-sleep signal between autonomic agents when needed, and an ALice signal to facilitate self-identification and self-certification between anonymous autonomous agents and systems.

1. Introduction

The field of Biology changed dramatically in 1953, with the determination by Francis Crick and James Dewey Watson of the double helix structure of DNA. This discovery changed Biology for ever, allowing the sequencing of the human genome, and the emergence of a "new Biology" focused on DNA, genes, proteins, data, and search. Computational Biology and Bioinformatics heavily rely on computing to facilitate research into life and development.

Simultaneously, an understanding of the biology of living organisms indicates a parallel with computing systems: molecules in living cells interact, grow, and transform according to the "program" dictated by DNA.

Moreover, paradigms of Computing are emerging based on modeling and developing computer-based systems exploiting ideas that are observed in nature. This includes building self-management and self-governance mechanisms that are inspired by the human body's autonomic nervous system into computer systems, modeling evolutionary systems analogous to colonies of ants or other insects, and developing highly-efficient and highly-complex distributed systems from large numbers of (often quite simple) largely homogeneous components to reflect the behaviour of flocks of birds, swarms of bees, herds of animals, or schools of fish.

This new field of "Biologically-Inspired Computing", often known in other incarnations by other names, such as: Autonomic Computing, Pervasive Computing, Organic Computing, Biomimetics, and Artificial Life, amongst others, is poised at the intersection of Computer Science, Engineering, Mathematics, and the Life Sciences. Successes have been reported in the fields of drug discovery, data communications, computer animation, control and command, exploration systems for space, undersea, and harsh environments, to name but a few, and augur much promise for future progress.

2. Safety and Security in Biologically-Inspired Systems

It is often joked that researchers in the security domain view safety as being a subset of security, while researchers in the safety domain see security as being a special case of safety. In fact, there is a certain degree of truth in both views, and valid cases can be made to support either position.

It is certainly true that various techniques from reliability engineering, safety engineering, and related areas, can be adapted to address issues in security. Similarly, protocols, analysis mechanisms, and other techniques from the security domain have been demonstrated to have useful application in safety-critical systems.

The classes of system that we're concerned with in this paper have their own particular issues vis-à-vis security and safety, however. Such systems are evolving, and adapting to the circumstances in their environment. More importantly, these systems are self-directed — we cannot necessarily tell *a priori* what situations they will be expected to address, nor necessarily what actions they will take to address them.

The FAST (Formal Approaches to Swarm Technologies) project looked at deriving a formal development method for swarm-based systems, a particular class of biologically-inspired system where (usually) a large number of components (whether software or physical devices) collaborate to achieve a common goal [22, 23]. As its example "test-bed", FAST used the ANTS (Autonomous Nano-Technology Swarm) concept mission, described in more detail in Section 4.

The project found (unsurprisingly) that no single formal development notation was sufficient to address all of the issues (in the case of ANTS, these were primarily safety-related issues, although security is not entirely discounted and likely to be a more important issue in actual operation). Moreover, it found that a realistic formal approach would require the use of a notation that made some allowance for the expression of probabilities and frequencies of operations. To this end, the FAST project proposed the combination of several formal notations, one of which is a probabilistic variant of a popular process algebra. The interested reader is directed to [26] for further details.

Further investigation, however, highlighted the fact that most of these probabilities and frequencies would be little more than guesswork, with a lot of the probabilities being so tiny (unlikely) that their combination would result in so many combinations of operations for which the probably was so close to zero that they couldn't be distinguished. We believe that this is likely to be the case with other types of biologically-inspired systems also. We simply do not have the experience to realistically estimate probabilities, nor are we ever likely to, since such systems are expected to "learn" and improve their operation over time.

As a result, any approach to the development of such systems (whether formal or otherwise) will be limited. That is not to say that there are not benefits from the use of formal approaches. In fact, FAST demonstrates that properties (safety properties, security properties, and others) may be proposed and proven to hold (or otherwise), giving certain degrees of assurance as to how the system will operate under *certain* conditions. Such an approach also allows for a significant amount of "what-if" analysis, where conditions can be formulated and in many cases it can be demonstrated that the system will be able to avoid, or if necessary, recover from, these conditions.

The reality, however, is that we cannot possibly foresee *all* such conditions and eventualities, and biologically-inspired systems must, as a consequence, have a greater number of prevention mechanisms built in, in order to ensure correct, safe, and secure operation.

3. Biologically-Inspired Computing Concepts

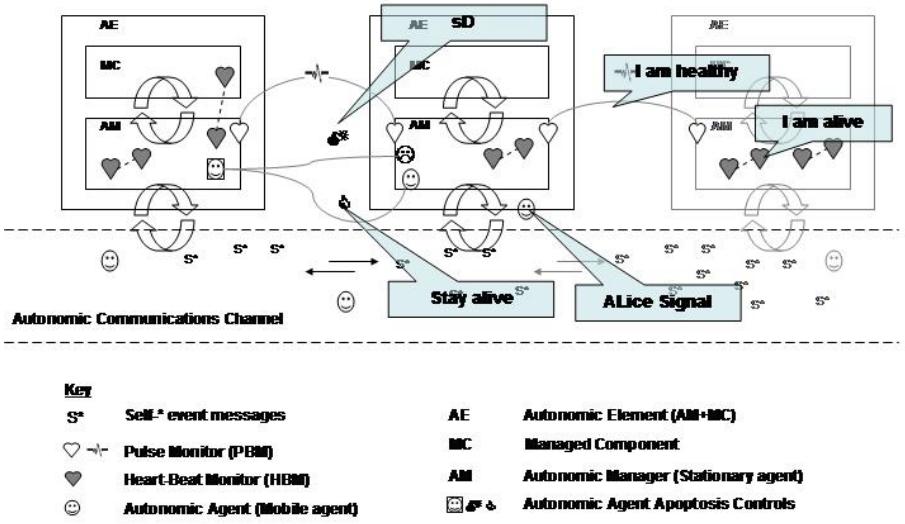


Figure 1 Autonomic and Autonomous Computing Environment

3.1 Autonomic Computing and Agents

Autonomic Computing is dependent on many disciplines for its success; not least of these is research in agent technologies. At this stage, there are no assumptions that agents have to be used in an autonomic architecture, but as in complex systems there are arguments for designing the system with agents [1], as well as providing inbuilt redundancy and greater robustness [2], through to retrofitting legacy systems with autonomic capabilities that may benefit from an agent approach [3] to research depicting the autonomic manager as an agent itself, for instance, a self-managing cell (SMC) [4], containing functionality for measurement and event correlation and support for policy-based control.

Figure 1 represents a view of an architecture for self-managing systems, where an autonomic element consists of the component required to be managed, and the autonomic manager [10]. It is assumed that an autonomic manager (AM) is responsible for a managed component (MC) within a self-contained autonomic element (AE). This autonomic manager may be designed as part of the component or provided externally to the component, as an agent, for instance. Interaction will occur with remote autonomic managers (cf. the autonomic communications channel shown in Figure 1) through virtual, peer-to-peer, client-server or grid configurations. The figure depicts self-* event messages as well as mobile agents, which assist with self-managing activity, traveling along this channel.

Essentially, the aim of autonomic computing is to create robust dependable self-managing systems [5]. To facilitate this aim, fault-tolerant mechanisms such as a heart-beat monitor ('I am alive' signals) and pulse monitor (urgency/reflex signals) may be included within the autonomic element [6, 7]. The notion behind the pulse monitor (PBM) is to provide an early warning of an undesirable condition so that preparations can be made to handle the processing load of diagnosis and planning a response, including diversion of load. Together with other forms of communications

it creates dynamics of autonomic responses [8] – the introduction of multiple loops of control, some slow and precise, others fast and possibly imprecise, fitting with the biological metaphors of reflex and healing [6].

3.2 Biological Apoptosis

The biological analogy of autonomic systems has been well discussed in the literature. While reading this, the reader is not consciously concerned with their breathing rate or how fast their heart is beating. Achieving the development of a computer system that can self-manage without the conscious effort of the user is the overarching vision of the Autonomic Computing initiative [9]. Another typical biological example is that the touching of a sharp knife results in a reflex reaction to reconfigure the area in danger to a state that is no longer in danger (self-protection, self-configuration, and, if damage has occurred, self-healing) [10].

If you cut yourself and it starts bleeding, you will treat it and carry on with your tasks without any further conscious thought. Yet, often, the cut will have caused skin cells to be displaced down into muscle tissue [11]. If they survive and divide, they have the potential to grow into a tumor. The body's solution to dealing with this situation is cell self-destruction. There is mounting evidence that some forms of cancer are the result of cells not dying fast enough, rather than multiplying out of control.

It is believed that a cell knows when to commit suicide because cells are programmed to do so – self-destruct (sD) is an intrinsic property. This self-destruction is delayed due to the continuous receipt of biochemical reprieves. This process is referred to as apoptosis [12], meaning "drop out", and was used by the Greeks to refer to the Autumn dropping of leaves from trees; i.e., loss of cells that ought to die in the midst of the living structure. The process has also been nicknamed "death by default" [13], where cells are prevented from putting an end to themselves due to constant receipt of biochemical "stay alive" signals.

Further investigations into the apoptosis process [14] have uncovered more details about this self-destruct predisposition. Whenever a cell divides, it simultaneously receives orders to kill itself. Without a reprieve signal, the cell does indeed self-destruct. It is believed that the reason for this is self-protection, as the most dangerous time for the body is when a cell divides, since if just one of the billions of cells locks into division the result is a tumor. However, simultaneously a cell must divide in order to build and maintain the body, and there is a constant conflict.

The suicide and reprieve controls have been compared to the dual-key on a nuclear missile [11]. The key (chemical signal) turns on cell growth but at the same time switches on a sequence that leads to self-destruction. The second key overrides the self-destruct [11].

3.3 The Role of Apoptosis within Autonomic Agents

Agent destruction has been proposed for mobile agents to facilitate security measures [15]. Greenberg et al. highlighted the situation simply by recalling the situation where the server omega.univ.edu was decommissioned, its work moving to other machines. When a few years later a new computer was assigned the old name, to the surprise of everyone, email arrived, much of it 3 years old [16]. The mail had survived "pending" on Internet relays waiting for omega.univ.edu to come back up.

Greenberg encourages consideration of the same situation for mobile agents; these would not be rogue mobile agents – they would be carrying proper authenticated credentials. This work would be done totally out-of-context due to

neither abnormal procedure nor system failure. In this circumstance the mobile agent could cause substantial damage, e.g., deliver an archaic upgrade to part of the network operating system resulting in bringing down the entire network.

Misuse involving mobile agents comes in the form of: misuse of hosts by agents, misuse of agents by hosts, and misuse of agents by other agents.

From an agent perspective, the first is through accidental or unintentional situations caused by that agent (race conditions and unexpected emergent behavior), the latter two through deliberate or accidental situations caused by external bodies acting upon the agent. The range of these situations and attacks have been categorized as: damage, denial-of-service, breach-of-privacy, harassment, social engineering, event-triggered attacks, and compound attacks.

In the situation where portions of an agent's binary image (e.g., monetary certificates, keys, information, etc.) are vulnerable to being copied when visiting a host, this can be prevented by encryption. Yet there has to be decryption in order to execute, which provides a window of vulnerability [16]. This situation has similar overtones to our previous discussion on biological apoptosis, where the body is at its most vulnerable during cell division.

The principles of a Hearth-Beat Monitor (HBM) and Pulse(-Beat) Monitor (PBM) have been established. Heart-Beat Monitor (I am alive) is a fault-tolerant mechanism which may be used to safeguard the autonomic manager, and to ensure that it is still functioning by periodically sending 'I am alive' signals. The Pulse Monitor (I am healthy) extends the HBM to incorporate reflex/urgency/health indicators from the autonomic manager, representing its view of the current self-management state. The analogy is with measuring the pulse rate to determine how healthy the patient is instead of merely detecting its existence (and the fact that the patient is alive).

Apoptosis (Stay alive) is a proposed additional construct used to safeguard both the system and agent; a signal indicates that the agent is still operating within the correct context and behavior, and should not self-destruct.

Is there a role for the apoptosis metaphor in the development of autonomic agents? [17, 18]

With many security issues, the lack of an agreed standard approach to agent-based systems prohibits, for now, further practical development of the use of apoptosis for agent security in a generic fashion within autonomic systems. Later, in a subsequent section, we propose a certification means between agents and hosts to work around this.

3.4 Autonomic Reflex Signal – Lub-Dub Pulse Emission

The autonomic environment requires that autonomic elements and, in particular, autonomic managers communicate with one another concerning self-* activities, in order to ensure the robustness of the environment. Figure 1 illustrates that the autonomic manager communications (AM⇔AM) also includes a reflex signal. This may be facilitated through the additional concept of a pulse monitor—PBM (an extension of the embedded system's heart-beat monitor, or HBM, which safeguards vital processes through the emission of a regular 'I am alive' signal to another process, as previously described) with the capability to encode health and urgency signals as a pulse [10]. Together with the standard event messages on the autonomic communications channel, this provides dynamics within autonomic responses and multiple loops of control, such as reflex reactions among the autonomic managers [20].

This reflex component may be used to safeguard the autonomic element by communicating its health to another AE [20]. The component may also be utilized to communicate environmental health information [10]. For instance, in the situation where each PC in a LAN is equipped with an autonomic manager, rather than each

of the individual PCs monitoring the same environment, a few PCs (likely the least busy machines) may take on this role and alert the others through a change in pulse rate to indicate changing circumstances.

An important aspect concerning the reflex reaction and the pulse monitor is the minimization of data sent – essentially only a "signal" is transmitted. Strictly speaking, this is not mandatory; more information may be sent, yet the additional information must not compromise the reflex reaction. For instance, in the absence of bandwidth concerns, information that can be acted upon quickly and not incur processing delays could be sent. The important aspect is that the information must be in a form that can be acted upon immediately and not involve processing delays (such as is the case of event correlation).

Just as the beat of the heart has a double beat (lub-dub) the autonomic element's (Figure 1) pulse monitor may have a double beat encoded – as described above, a *self* health/urgency measure and an *environment* health/urgency measure. These match directly with the two control loops within the AE, and the self-awareness and environment awareness properties.

3.5 The ALice Signal

An aspect to this research is that Anonymous Autonomous/Autonomic Agents need to work within the Autonomic System to facilitate self-management; as such the agents and their hosts need to be able to identify each other's credentials through such means as an ALice (Autonomic License) signal [19]. This would allow a set of communications to ensure the visiting mobile agent has valid and justified reasons for being there as well as providing security to the visiting agent in interaction with other agents and host. An unsatisfactory ALice exchange may lead to self-destruction for self-protection.

3.6 Biological Quiescence

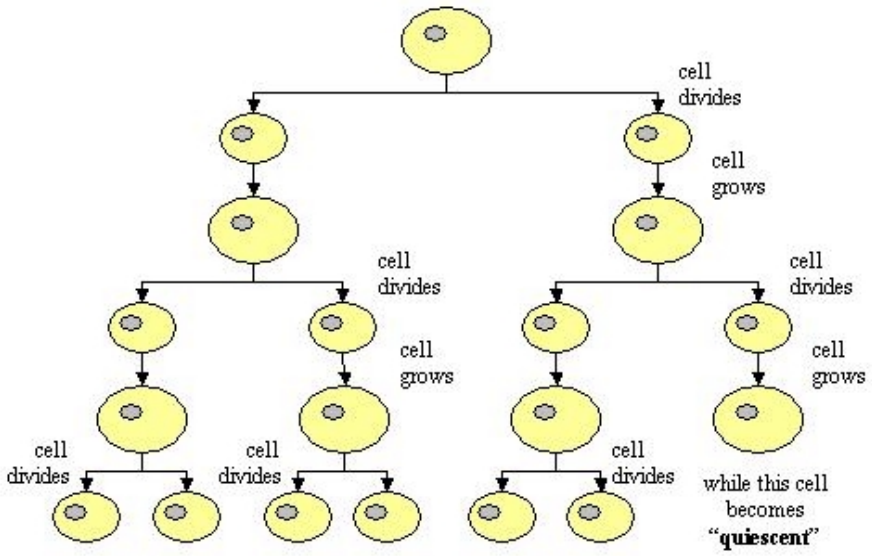


Figure 2 Biological Cell Cycle - including Quiescent Cell

The biological cell cycle is often described as a circle of cell life and division. A cell divides into two "daughter cells" and both of these cells live, "eat", grow, copy their genetic material and divide again producing two more daughter cells. Since each daughter cell has a copy of the same genes in its nucleus, daughter cells are "clones" of each other. This "twinning" goes on and on with each cell cycle. This is a natural process.

Very fast cell cycles occur during development causing a single cell to make many copies of itself as it grows and differentiates into an embryo. Some very fast cell cycles also occur in adult animals. Hair, skin and gut cells have very fast cell cycles to replace cells that die naturally. While, as was highlighted earlier, some forms of cancer may be caused by cells cycling out of control (as well as not dying quickly enough).

But there is a kind of "parking spot" in the cell cycle, called "quiescence". A **quiescent** cell has left the cell cycle, it has stopped dividing. Quiescent cells may re-enter the cell cycle at some later time, or they may not; it depends on the type of cell. Most nerve cells stay quiescent forever. On the other hand, some quiescent cells may later re-enter the cell cycle in order to create more cells (for example, during pubescent development) [21].

3.7 The Role of Quiescence within Autonomic Agents

The agent self-destruction proposed earlier (Autonomic Apoptosis) to facilitate security measures may be considered an extreme or ultimate self-protection measure – for cases when the agent's security has been breached or the agent is endangering the system (for instance demonstrating undesirable emergent behavior) [17, 18]. Yet, not all cases may require this extreme reaction. Self-sleep (Quiescent state) instead of self-destruct (Apoptosis) may be all that is required for certain circumstances. As the situation emerges and is clarified, the agent may resume its activity or be put into an apoptotic state.

In the case of Greenberg's authenticated mobile agent carrying an archaic upgrade, as described in Section 3.3, since this is a about to perform an activity that poses a security risk, its intrinsic nature could be such that it enters a quiescent state until its behavior is confirmed and before it proceeds with its activity. As was highlighted earlier, these situations have similar overtones to where the body is at its most vulnerable during cell division. High-risk security self-managing activity can be protected by apoptosis and quiescence used to act as intrinsic mechanisms for self-destruct or self-sleep.

4. Biologically-Inspired Concepts and Autonomicity for future NASA Missions

These concepts may assist in the new radical paradigms for spacecraft design to facilitate adaptive operations and the move towards almost total onboard autonomy in certain classes of mission operations [22, 23].

A concept mission, ANTS, Autonomous Nano-Technology Swarm, planned for sometime between 2020 and 2030 is viewed as a prototype for how many future unmanned missions will be developed and how future space exploration will exploit autonomous and autonomic behavior.

The mission will involve the launch of 1000 pico-class spacecraft swarm from a stationary factory ship, on which the spacecraft will be assembled. The spacecraft will explore the asteroid belt from close-up, something that cannot be done with conventionally-sized spacecraft.

As much as 60% to 70% of the spacecraft will be lost on first launch as they enter the asteroid belt. The surviving craft will work as a swarm, forming smaller groupings of *worker* craft (each containing a unique instrument for data gathering), a coordinating *ruler*, that will use the data it receives from workers to determine which asteroids are of interest and to issue instructions to the workers and act as a coordinator, and *messenger* craft which will coordinate communications between the swarm and between the swarm and ground control. Communications with earth will be limited to the download of science data and status information, and requests for additional craft to be launched from earth as necessary.

Section 2 clearly highlights the general problem of agent security, whether from the agent's or host's perspective. In terms of generic contribution to autonomic agent development, with many security issues the lack of an agreed standard approach to agent-based systems prohibits immediate further practical development of apoptosis and quiescent states for generic autonomic systems.

Of course, within NASA missions, such as ANTS, we are not considering the generic situation. Mission control and operations is a trusted private environment. This eliminates many of the wide range of agent security issues discussed earlier, just leaving the particular concerns: is the agent operating in the correct context and showing emergent behavior within acceptable parameters, where upon *apoptosis* and *quiescence* can make a contribution?

For instance, in ANTS, suppose one of the *worker* agents was indicating incorrect operation, or when co-existing with other workers was the cause of undesirable emergent behavior, and was failing to self-heal correctly. That emergent behavior (depending on what it was) may put the scientific mission in danger. The agent may be put to sleep or ultimately the stay alive signal from the *ruler* agent would be withdrawn.

If a *worker*, or its instrument, were damaged, either by collision with another worker, or (more likely) with an asteroid, or during a solar storm, a *ruler* could withdraw the stay alive signal and request a replacement *worker* (from Earth, if necessary). If a *ruler* or *messenger* were similarly damaged, its stay alive signal would also be withdrawn, and a *worker* would be promoted to play its role. During a solar storm the *workers* could be put into a quiescent state to protect themselves from damage.

All of the spacecraft are to be powered by batteries that are recharged by the sun using solar sails [22, 23]. Although battery technology has greatly advanced, there is still a "memory loss" situation, whereby batteries that are continuously recharged eventually lose some of their power and cannot be recharged to full power. After several months of continual operation, each of the ANTS will no longer be able to recharge sufficiently, at which point their 'stay alive' signals will be withdrawn, and new craft will need to be assembled or launched from Earth.

5. Related Work

Forrest *et al*. [27] in their classic work described the problem of protecting computer systems as a general problem of learning to distinguish *self* (legitimate users, corrupted data, etc.) from *other* (unauthorized users, viruses, etc.); their solution was a method for change detection inspired by the generation of T cells in the immune system [28].

In relation to the Autonomic Initiative, the autonomic manager may take on this function of self-/non-self discrimination as part of its self-awareness in order to facilitate self-protection. Yet to achieve the envisaged Autonomic Initiative long-term vision of system-level self-direction and self-management requires a high level of interaction among AMs; and since AMs at the local level will view their world as *self,* activity from the external environment may be perceived from a local AM view

as *others/non-selfs*. (In the greater scheme of things, all these legitimate self-management activities will actually be *self* as opposed to *other/non-self* but the sheer vastness of systems of systems could result in a local AM perception/classification that these legitimate activities are of *other/non-self*). As such, the work described in this paper is complementary to Forrest *et al*'s research. In our approach, the ALice concept is used to identify and distinguish agents from the external environment, indeed part of the greater *self* as opposed to *other/non-self.* Additionally, complementary biological inspiration is derived from apoptosis and quiescence for intrinsic mechanisms to facilitate correct operation by *self* (for instance avoiding undesirable emergent behavior) and not just to distinguish *self* from *non-self/other*.

6. Conclusions

Autonomic agents have been gaining ground as a significant approach to facilitate the creation of self-managing systems to deal with the ever increasing complexity and costs inherent in today's and tomorrow's systems.

In terms of the Autonomic Systems initiative, agent technologies have the potential to become an intrinsic approach within the initiative [24], not only as an enabler (e.g., ABLE agent toolkit [25]), but also in terms of creating autonomic agent environments.

Apoptosis was introduced and previously discussed in [17]. We have extended this here with Autonomic Quiescence—self-sleep, a less drastic form of self-protection.

We have briefly described research into biologically-inspired concepts to be used together as intrinsic mechanisms within agents to provide inherent safety and security both at the agent and system level. We briefly discussed this in terms of the NASA concept mission, ANTS. More detailed accounts of the ANTS mission are given in [23] and [26]. We continue to seek inspiration for modeling and developing computer-based systems from ideas that are observed in nature.

Acknowledgements

This research is partly supported at University of Ulster by the Computer Science Research Institute and the Centre for Software Process Technologies (CSPT) which is funded by Invest NI through the Centres of Excellence Programme, under the EU Peace II initiative.

Part of this work has been supported by the NASA Office of Systems and Mission Assurance (OSMA) through its Software Assurance Research Program (SARP) project, Formal Approaches to Swarm Technologies (FAST), and by NASA Goddard Space Flight Center, Software Engineering Laboratory (Code 581).

Some of the technologies described in this article are patent pending and assigned to the United States government.

References

1. N.R. Jennings and M. Wooldridge, "Agent-oriented Software Engineering," in J. Bradshaw (ed.), *Handbook of Agent Technology*, AAAI/MIT Press, 2000.
2. M..N. Huhns, V.T. Holderfield and R.L.Z. Gutierrez, "Robust software via agent-based redundancy," In *Proc. Second International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2003*, 14-18 July 2003, Melbourne, Victoria, Australia, pp. 1018-1019.

3. G. Kaiser, J. Parekh, P. Gross, G. Valetto, "Kinesthetics eXtreme: An External Infrastructure for Monitoring Distributed Legacy Systems," *In Proc. Autonomic Computing Workshop – IEEE Fifth Annual International Active Middleware Workshop*, Seattle, WA, USA, June 2003.
4. E. Lupu *et al*., EPSRC AMUSE: Autonomic Management of Ubiquitous Systems for e-Health, 2003.
5. R. Sterritt, D.W. Bustard, "Autonomic Computing: a Means of Achieving Dependability?" In *Proceedings of IEEE International Conference on the Engineering of Computer Based Systems (ECBS'03)*, Huntsville, AL, USA, 7-11 April 2003, pp. 247-251.
6. R. Sterritt, "Pulse Monitoring: Extending the Health-check for the Autonomic GRID," In *Proceedings of IEEE Workshop on Autonomic Computing Principles and Architectures (AUCOPA 2003) at INDIN 2003*, Banff, AB, Canada, 22-23 August 2003, pp. 433-440.
7. R. Sterritt, "Towards Autonomic Computing: Effective Event Management," In *Proceedings of 27th Annual IEEE/NASA Software Engineering Workshop (SEW)*, Maryland, USA, 3-5 December 2002, IEEE Computer Society Press, pp. 40-47.
8. R. Sterritt, D.F. Bantz, "PAC-MEN: Personal Autonomic Computing Monitoring Environments," *In Proceedings of IEEE DEXA 2004 Workshops - 2nd International Workshop on Self-Adaptive and Autonomic Computing Systems (SAACS 04),* Zaragoza, Spain, 30 August – 3 September, 2003.
9. J. O. Kephart and D. M. Chess. "The vision of autonomic computing". *Computer*, 36(1):41–52, 2003.
10. R. Sterritt, D.W. Bustard, "Towards an Autonomic Computing Environment," In *Proceedings of IEEE DEXA 2003 Workshops - 1st International Workshop on Autonomic Computing Systems*, Prague, Czech Republic, September 1-5, 2003, pp. 694-698.
11. J. Newell, "Dying to live: why our cells self-destruct," Focus, Dec. 1994.
12. R. Lockshin, Z. Zakeri, "Programmed cell death and apoptosis: origins of the theory," *Nature Reviews Molecular Cell Biology*, 2:542-550, 2001.
13. Y. Ishizaki, L. Cheng, A.W. Mudge, M.C. Raff, "Programmed cell death by default in embryonic cells, fibroblasts, and cancer cells," *Mol. Biol. Cell*, 6(11):1443-1458, 1995.
14. J. Klefstrom, E.W. Verschuren, G.I. Evan, "c-Myc Augments the Apoptotic Activity of Cytosolic Death Receptor Signaling Proteins by Engaging the Mitochondrial Apoptotic Pathway," *J Biol Chem*,. 277:43224-43232, 2002.
15. J.D. Hartline, *Mobile Agents: A Survey of Fault Tolerance and Security*, University of Washington, 1998.
16. M.S. Greenberg, J.C. Byington, T. Holding, D.G. Harper, "Mobile Agents and Security," *IEEE Communications*, July 1998.
17. R. Sterritt and M.G. Hinchey, "Apoptosis and Self-Destruct: A Contribution to Autonomic Agents?" In *Proceedings FAABS-III, 3rd NASA/IEEE Workshop on Formal Approaches to Agent-Based Systems*, 26-27 April 2004, Greenbelt, MD, Springer Verlag LNCS 3228, 2005.
18. R. Sterritt and M.G. Hinchey, "Engineering Ultimate Self-Protection in Autonomic Agents for Space Exploration Missions," In *Proceedings of IEEE Workshop on the Engineering of Autonomic Systems (EASe 2005) at 12th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2005)*, Greenbelt, MD, USA, 3-8 April 2005, IEEE Computer Society Press, pp. 506-511.
19. R. Sterritt and M.G. Hinchey, "Biological Inspired Concepts for Self-Managing Ubiquitous and Pervasive Computing Environments" In *Proceedings WRAC-II, 2nd NASA/IEEE Workshop on Radical Agent Concepts*, Sept. 2005, Greenbelt, MD, Springer Verlag LNCS, 2006.
20. R. Sterritt, D. Gunning, A. Meban, and P. Henning, Exploring Autonomic Options in a Unified Fault Management Architecture through Reflex Reactions via Pulse Monitoring. Proceedings of IEEE Workshop on the Engineering of Autonomic Systems (EASe 2004) at the 11th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS 2004), Brno, Czech Republic, 24–27 May, pp 449–455
21. J. Love, *Science Explained,* 1999.

22. M.G. Hinchey, J.L. Rash, W.F. Truszkowski, C.A. Rouff and R. Sterritt, "Challenges of Developing New Classes of NASA Self-Managing Missions," In *Proceedings of Workshop on Reliability and Autonomic Management in Parallel and Distributed Systems (RAMPDS-05) at ICPADS-2005*, Fukuoka, Japan, 20-22 July 2005, pp. 463-467.
23. M.G. Hinchey, J.L. Rash, W.F. Truszkowski, C.A. Rouff and R. Sterritt, "Autonomous and Autonomic Swarms," *In Proceedings of Autonomic & Autonomous Space Exploration Systems (A&A-SES-1) at 2005 International Conference on Software Engineering Research and Practice (SERP'05),* Las Vegas, NV, 27-30 June 2005, CREA Press, pp. 36-42.
24. J. McCann and M. Huebscher, "Evaluation issues in Autonomic Computing," In H. Jin, Y. Pan, N. Xiao, and J. Sun (Eds.): GCC 2004 Workshops, LNCS 3252, pp. 597–608, 2004.
25. J.P. Bigus *et.al*., "ABLE: a toolkit for building multiagent autonomic systems," *IBM Systems J.*, **41**(3):350-371, 2002.
26. C.A. Rouff, M.G. Hinchey, W.F Truszkowski, and J.L. Rash, "Experiences Applying Formal Approaches in the Development of Swarm-Based Space Exploration Missions," *International Journal of Software Tools for Technology Transfer*, 2006, to appear.
27. S. Forrest, A.S. Perelson, L. Allen, R. Cherukuri, "Self-nonself discrimination in a computer," In *Proc. IEEE Symposium on Research in Security and Privacy 1994*, 16-18 May 1994, pp. 202-212.
28. P. D'haeseleer, S. Forrest, P. Helman, "An immunological approach to change detection: algorithms, analysis and implications," In *Proc. IEEE Symposium on Security and Privacy, 1996*, 6-8 May 1996, pp. 110-119.

Article submission for *IEEE Computer*, Standard's Column, August 2006
(Article, with technical editing by IEEE did appear in the Aug 2006 issue)

Toward a Unified Safety/Security Model

The worlds of safety and security have co-existed for some time, yet remain largely separate domains with limited interactions. This is, to put it mildly, a problem. Each domain has contributions for the other and more dependable systems being a significant benefit of working together. Yet one element that has continued to separate these domains is lack of a common language and taxonomy for discussing risks associated with safety and with security. This articles proposes a common risk taxonomy framework that bridges the existing security and safety risk frameworks; producing one common expression readily applicable in either domain.

Cooperation between safety and security communities enable mutual support toward the common goal of achieving systems that are dependable in a real world where they are exposed to events that can result in harm. This includes harm to:

- Individuals, be it health and life (safety) or loss of privacy, monetary loss, or harm to image or reputation (security issues),
- Assets (generally a security issue yet also a safety issue), and
- Organizations; for example, an organization's mission, function, image, or reputation (security issues).

Safety and security share common concerns within the list of potential harmful events above. Specifically, the security events that can result in loss of life or health are clearly also safety concerns. Moreover, the processes in place for addressing safety presume a probabilistic distribution for failures that could lead to safety. This is a fundamental, underlying assumption for all safety processes that does not hold for intelligent attacks. Hence cooperation between the domains of safety and security is essential for safety to be achieved in the real world of high dependence on information systems subject to attack by malicious, capable adversaries.

The domain of security can benefit at least as much from cooperation. In many instances organizations have inculcated a culture of safety-consciousness that truly impacts the decisions being made. That cannot be said as often for security, where the functional gains to be achieved by automation frequently overwhelm concerns about risks to individuals, the organization, and its assets. Too often the question of 'how much security is enough?' is answered by how much effort has been expended with seemingly little regard for the remaining risks arising from the automation of mission/business processes. Wherever a safety culture already exists, bringing common expression to the safety and security risks can be expected to enhance understanding of the security issues and facilitate security/function tradeoffs being done in a manner consistent with that already being implement for safety issues.

A generally accepted risk taxonomy for security can be found in the U.S. Federal

guidance on risk management in National Institute of Standards and Technology (NIST) Special Publication 800-30, *Risk Management Guide for Information Technology Systems*, July 2002 (http://csrc.nist.gov/publications/nistpubs/800-30/sp800-30.pdf). The security taxonomy captured in this document is depicted in Figure 1.

A common risk taxonomy for the safety community is found in Federal Aviation Administration Order 8040.4, Appendix G, *Safety Risk Management*, June 1998 (http://www.faa.gov/library/manuals/aviation/risk_management/ss_handbook/media/app_g_1200.PDF). This taxonomy is shown in Figure 2.

While the two taxonomies are enough different to reinforce the separation between the two domains, there is a significant amount of potential overlap, requiring no significant change to the safety terminology other than providing for an extension to what types of events constitute hazards. A common taxonomy can be achieved simply by enlarging the safety term 'hazard' and adopting a unified definition for the term 'mishap'.

The current safety definition for 'hazard' is: "Condition, event, or circumstance that could lead to or contribute to an unplanned or undesired event." [FAA Order 8040.4]. This definition can be broad enough to also incorporate the security incidents classed as 'threats' in NIST SP 800-30; as these certainly are part of the 'condition, event, or circumstance' that leads to an 'undesired event'.

With regard to a 'mishap', the current definitions for mishap and threat are:

> Mishap: Unplanned event, or series of events, that results in death, injury, occupational illness, or damage to or loss of equipment or property. [FAA Order 8040.4]
>
> Threat: The potential for a threat-source to exercise (accidentally trigger or intentionally exploit) a specific vulnerability. [NIST SP 800-30]

A suggested unified definition for the term 'mishap' is:

> Mishap: Unified safety/security definition: Unplanned event, or series of events, that results in death, injury, occupational illness, or other harm to individuals well-being (to include privacy, finances, image, and reputation); damage to or loss of equipment or property; or harm to an organization (mission, function, image, or reputation). These events include system/equipment/component failures, requirement/design/implementation flaws, user errors, and intentional attacks.

With this unified definition for "mishap" and the incorporation of security incidents into the term 'hazard', a unified security/safety taxonomy is readily obtained as shown in Figure 3.

A common taxonomy for addressing risks is an important practical element of bringing the domains of safety and security together into an effective, mutually-supporting relationship. The proposed unified, risk taxonomy provides this common expression and

does so with limited change to existing taxonomies.

Both safety and security have much to gain by working together. Security can piggy-backing on the work done within the safety community in developing definitions and terminology to express hazard conditions and in establishing organizational awareness of need to trade function for other, important concerns. The safety community can take advantage of the work done by security in dealing with intelligent maliciousness which is not well addressed by the probabilistic assumptions that underlie safety processes and yet is now a significant concern with regard to safety. A risk framework has been proposed to help make the idea of working together more than just an idea, but a reality.

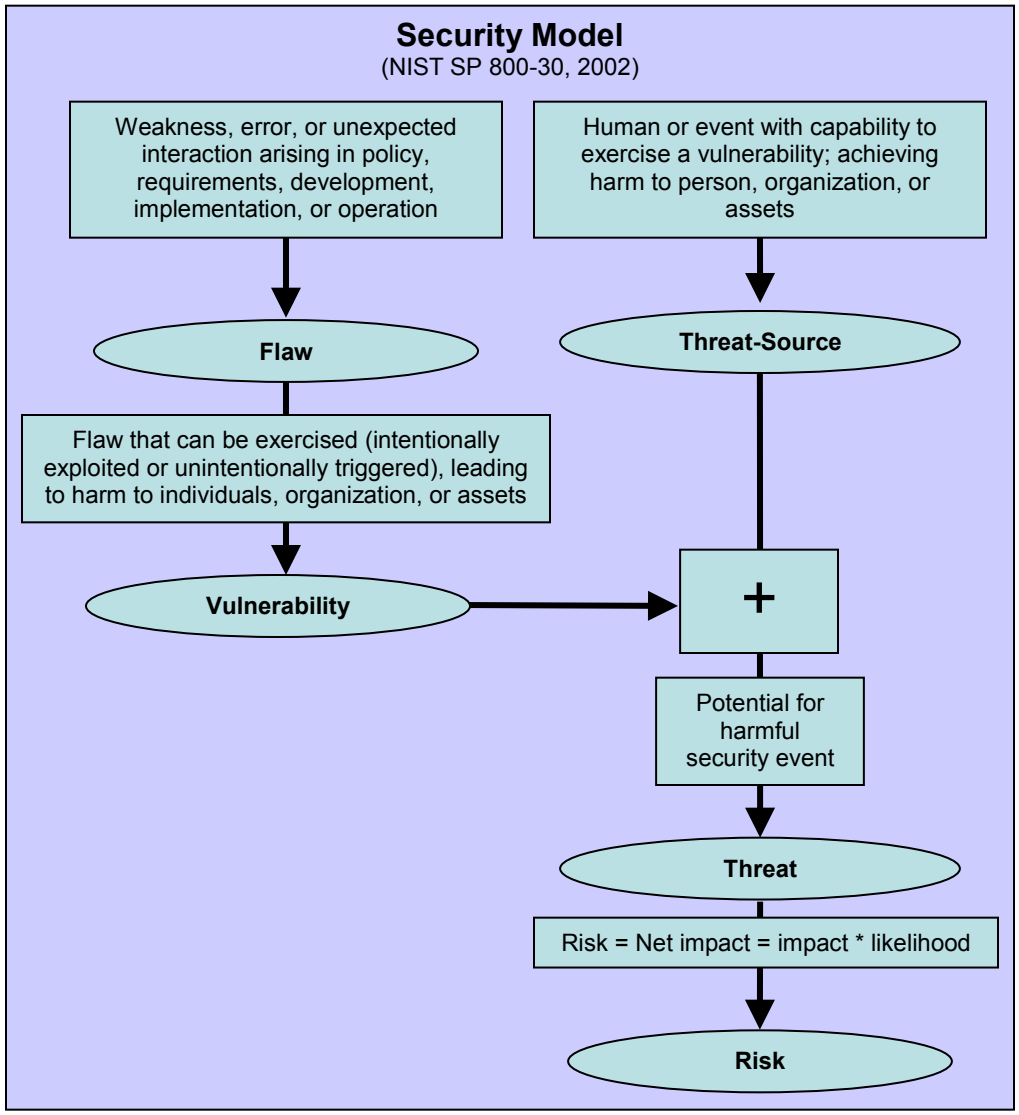


Figure 1: Security Risk Framework (NIST SP 800-30)

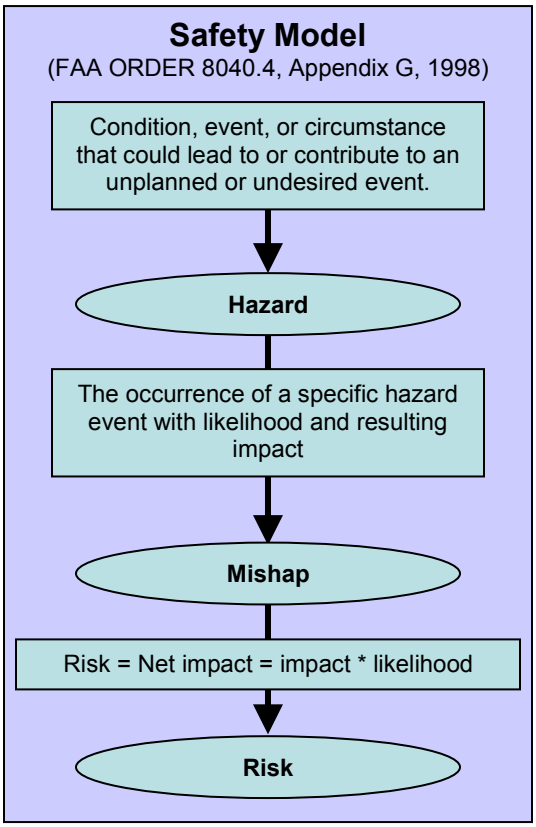


Figure 2: Safety Risk Framework (FAA ORDER 8040.4, Appendix G)

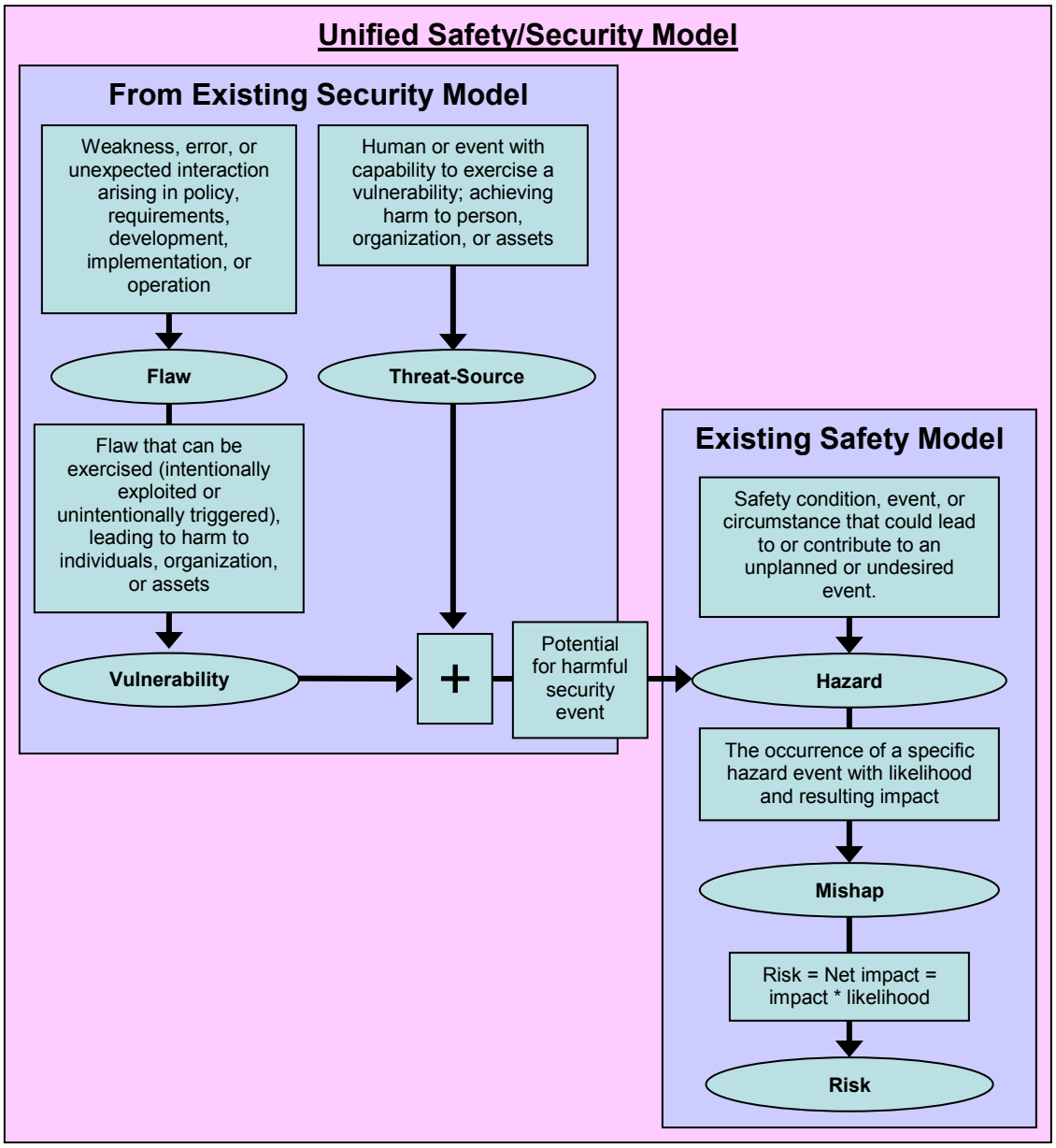


Figure 2: Unified Safety/Security Risk Framework

Author:
Gary Stoneburner
Johns Hopkins University/Applied Physics Laboratory (JHU/APL)
Gary.Stoneburner@jhuapl.edu
240-228-2628

Author Bio:
Gary Stoneburner is an electronic engineer with Master of Science in Electrical Engineering from the University of Texas (1974) and a Bachelor of Engineering Science from Johns Hopkins University (1972). He is a member of the senior professional staff at the Johns Hopkins University/Applied Physics Laboratory (JHU/APL) where he provides information system security engineering for DoD and civil federal agencies. He was previous with the National Institute of Standards and Technology (NIST) where he served as the technical advisor to the NIST FISMA Implementation project. Prior to coming to NIST he served as the security architect for The Boeing Company. Gary retired from the US Army Reserve in 2004 where his last assignment was with the Army Network Operations and Security Center (ANOCS). Previous reserve assignments include the Army's Information Operations Red Team; the Army Computer Emergency Response Team (ACERT); and Deputy Chief, Information Assurance Division, J6, USSOUTHCOM. He is the author or co-author for several NIST publications including: NISTIR 6462 *CSPP - Guidance for COTS Security Protection Profiles*, NISTIR 6985 *COTS Security Protection Profile - Operating Systems (CSPP-OS),* SP 800-27 *Engineering Principles for IT Security (EP-ITS)*, SP 800-30 *Risk Management Guideline*, SP 800-33 *Underlying Technical Models for IT Security*, SP 800-37 *Certification and Accreditation of Federal IS*, SP 800-53 Rev 1 *Recommended Security Controls for Federal Systems*.

Toward a Unified Safety/Security Model

IWSS-07
March 15, 2007
Gary.Stoneburner@jhuapl.edu
240-228-2628

Safety – Security Coexisting/Independent

- The worlds of safety and security have co-existed for some time
- Common goal of achieving systems that are dependable in a real world where they are exposed to events that can result in harm
- Yet remain largely separate domains with limited interactions.

This is, to put it mildly, a problem

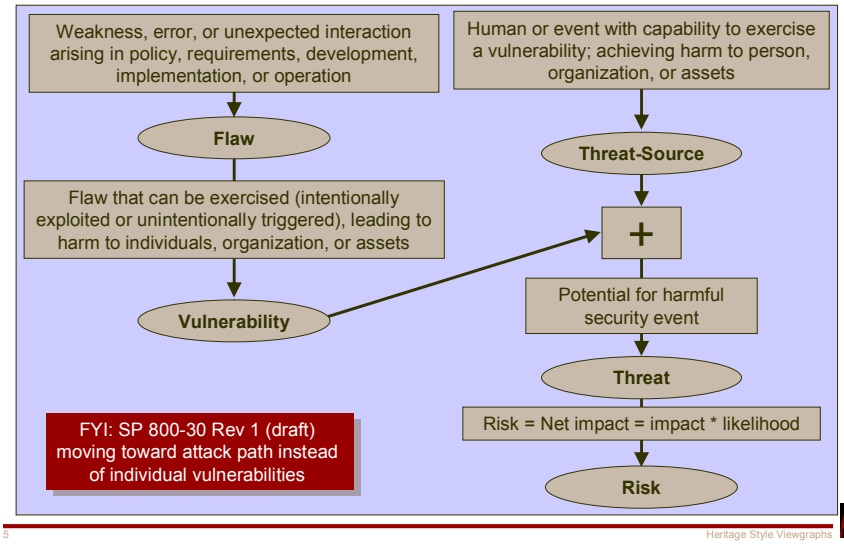
Proposal to help

- A common risk taxonomy framework
  - Bridge the existing security and safety risk frameworks
  - Common expression readily applicable in either domain.

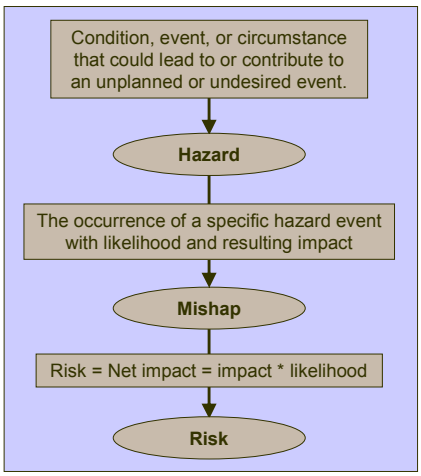
Value of a seeking a common taxonomy

- One element that has continued to separate safety and security domains is lack of a common language and taxonomy for discussing risks
- Facilitate mutual support toward the common goal of achieving systems that are dependable in a real world where they are exposed to events that can result in harm
- Each domain has contributions for the other and more dependable systems being a significant benefit of working together.
  - Safety benefits from security: Fundamental, underlying assumptions for safety processes do not hold for purposeful attacks – **security addresses purposeful maliciousness**
  - Security benefits from safety: Some organizations have inculcated a culture of safety-consciousness that truly impacts the decisions being made **– willingness to tradeoff function for system 'ility'**

Existing Security Taxonomy
NIST SP 800-30, 2002



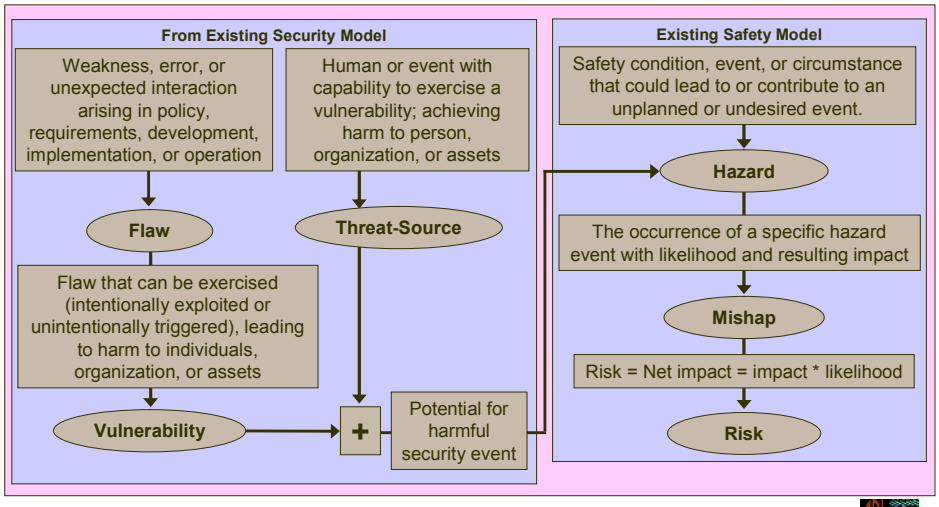
Existing Safety Taxonomy
FAA ORDER 8040.4, Appendix G, 1998



Terminology Suggestions

- **Hazard:** No change to text of existing safety definition, just incorporate into current understanding of safety 'undesired events' the events classed as 'threats' in NIST SP 800-30. [Rationale: These 'security' events certainly are part of the 'condition, event, or circumstance' that leads to an 'undesired' safety event.]
- **Mishap:** Unified safety/security definition: Unplanned event, or series of events, that results in (a) death, injury, occupational illness, or other harm to individuals well-being (to include privacy, finances, image, and reputation); (b) damage to or loss of equipment or property; or (c) harm to an organization (mission, function, image, or reputation). These events include failures, flaws, user errors, and intentional attacks.

Resulting Merged Taxonomy



SAIC
An Employee-Owned Company

Juggling With the Software Assurance Puzzle Pieces

Jeffrey Voas

Opinions expressed here are those solely of Jeffrey Voas, and not necessarily those of SAIC.

Talk Outline

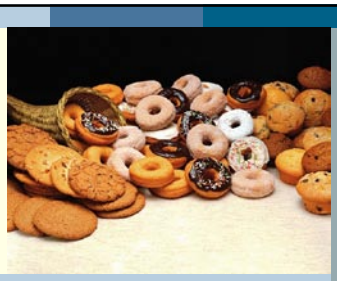
- I. Basic Principles Framing Software Assurance
- II. The Role of Standards and Certification in Software Assurance Puzzle
- III. The Technical and Financial Trade-off Issues of the Attribute “ilities”
- IV. A little slide presentation

Dispel the Myths That Say…

- 1.Software Assurance is Secure software
- 2. Software Assurance is Computer security
- 3. Software Assurance is Information assurance
- 4. Software Assurance is guaranteed using Standards
- 5. Software Assurance is Well-defined, Framed, and Bounded

Stress the Idea That …

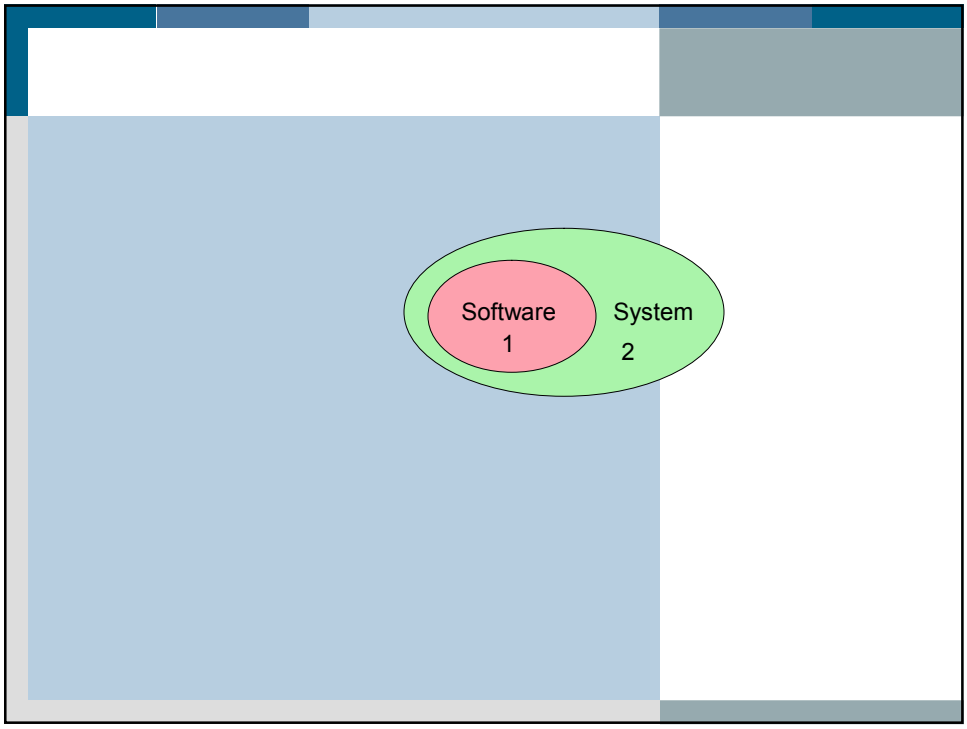
1. Software assurance is always a function of time,
2. Software today is less and less viewed as a product, it is more viewed as a service, and therefore service assurance and resilience are vital to contemplate,
3. Software assurance is a non-boundable problem, except in rare cases, such as embedded systems.

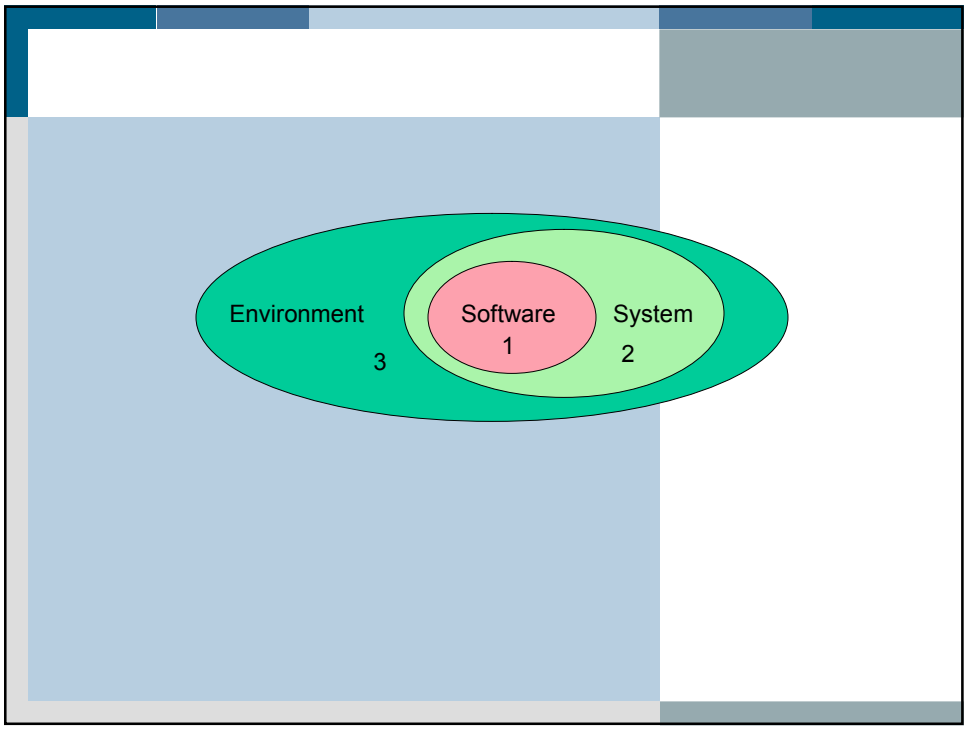


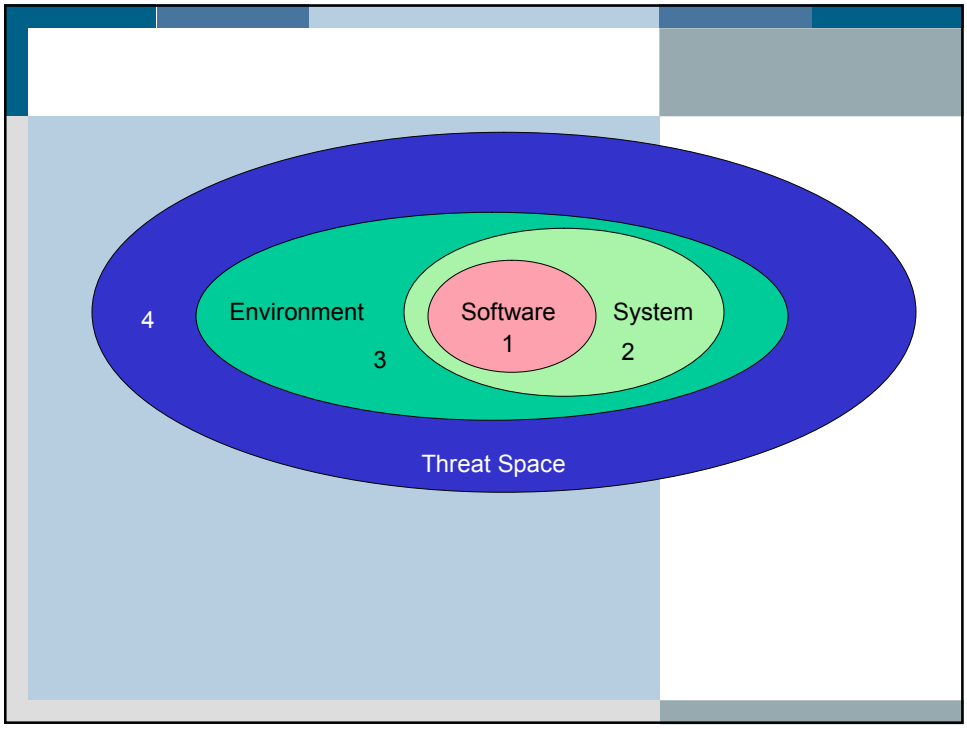
Well-defined, framed, and bounded?

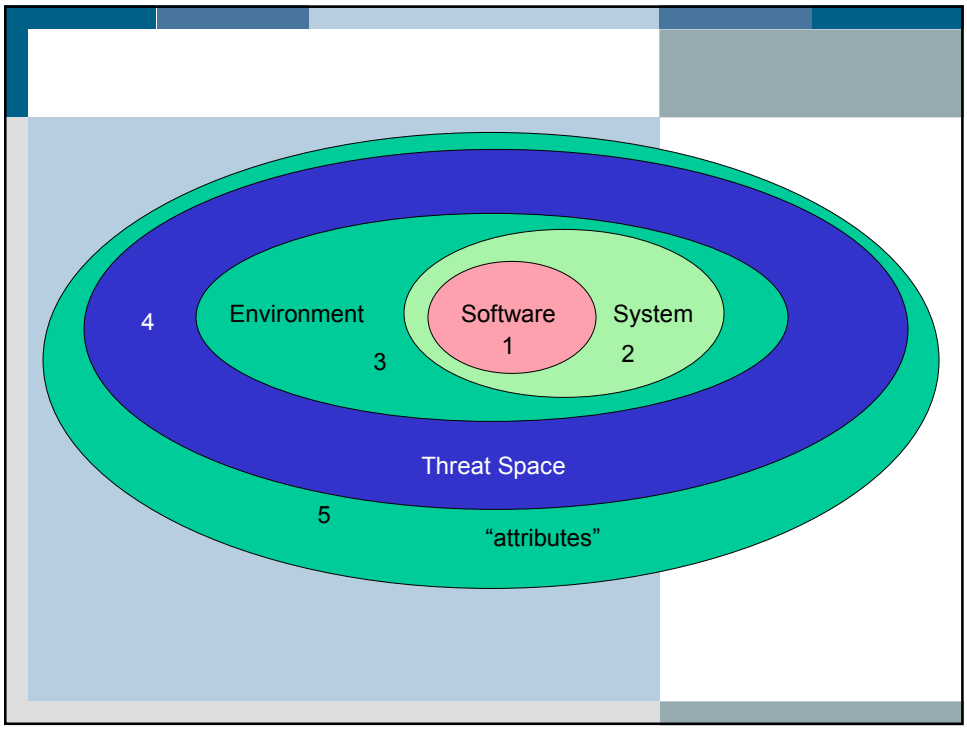
A Different Way Forward to Frame/Bound/Define

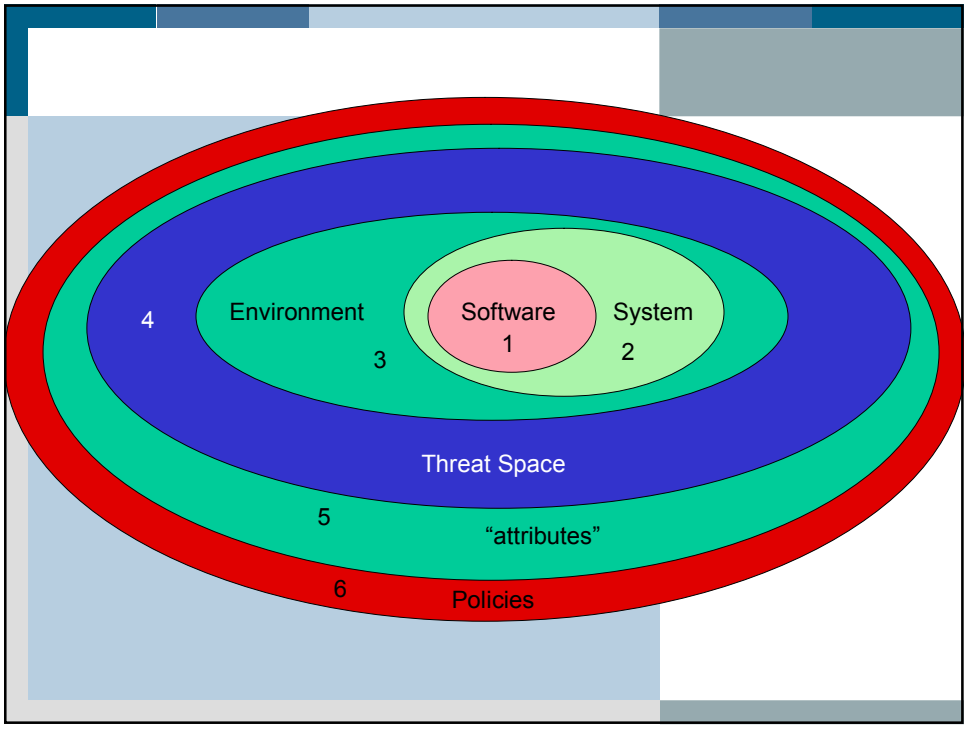
Software 1

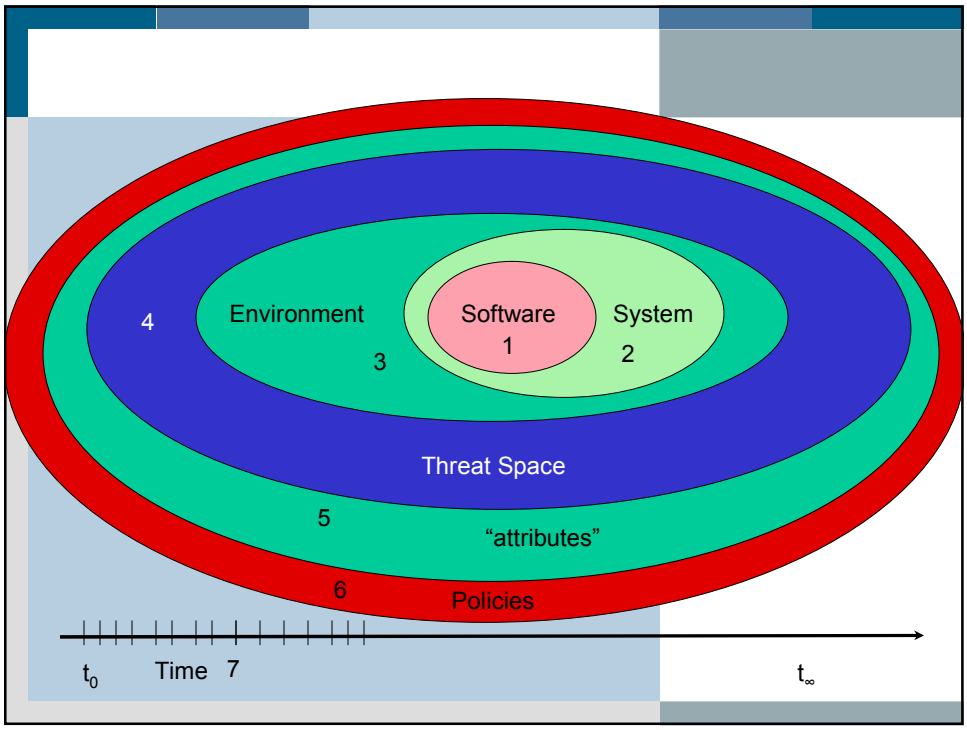

Software
1
System
2

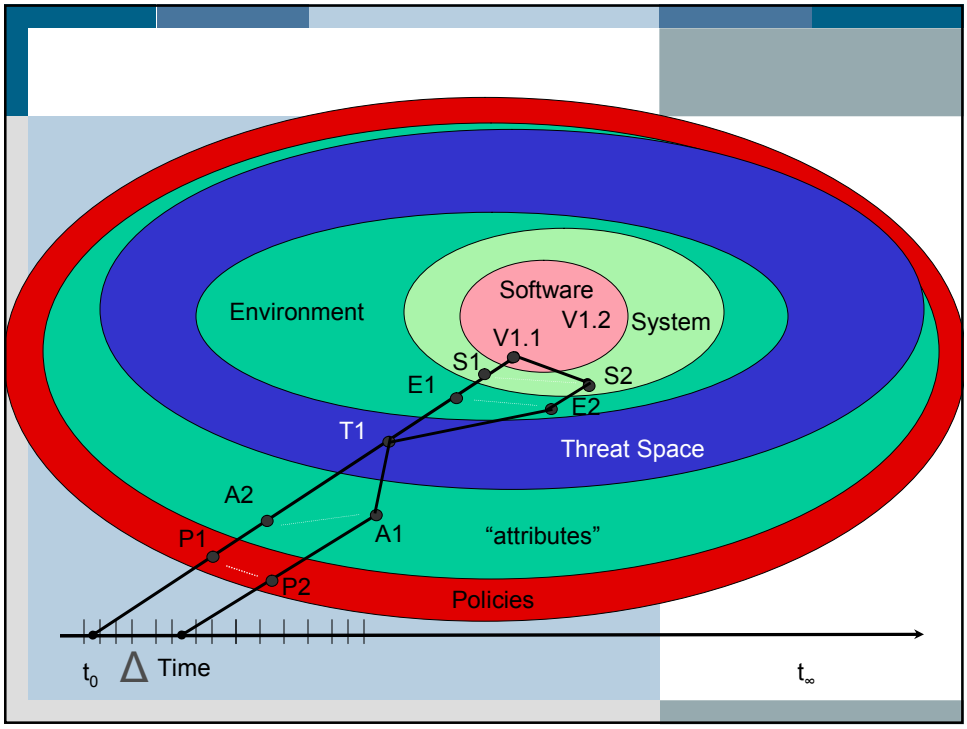

Environment
3
Software
1
System
2


4
Environment
3
Software
1
System
2
Threat Space

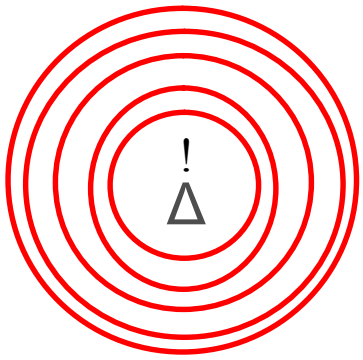

4
Environment
3
Software
1
System
2
Threat Space
5
“attributes”


4
Environment
Software
1
System
2
3
Threat Space
5
"attributes"
6
Policies


4
Environment
Software
1
System
2
3
Threat Space
5
"attributes"
6
Policies
t_0
Time
7
t_∞


Environment
Software
V1.2
System
V1.1
S1
S2
E1
E2
T1
Threat Space
A2
P1
A1
"attributes"
P2
Policies
t0
Δ Time
t∞

Bullseye


!
Δ

Software Engineering Standards for Assurance

Simply lines in the sand from which a certificate of compliance or non-compliance can occur.

Point 1: Standards and Certification are logically inseparable.

Point 2: If you cannot certify against a standard, why have a standard? Question: For hand waving?

Point 3: C&A

Three Key Messages That Certification Against A Standard Can Convey

- Compliance with development process standards *vs*.
- Fitness for purpose *vs*.
- Compliance with the requirements

Premise for Software Engineering Standards

Acquired software should be tagged with some guarantee (or at least a "warm fuzzy") as to how "good" the software is.

Problem: Software Of Unknown Pedigree (SOUP)

Goal of Software Engineering Standards: SOKP

Problem: How good is "good enough"?

State-of-the-Practice/Art

"A consumer [patient] may not be able to assess accurately whether a particular drug is safe, but [they] can be reasonably confident that drugs obtained from approved sources have the endorsement of the U.S. Food and Drug Administration (FDA) which confers important safety information. Computer system trustworthiness has nothing comparable to the FDA. **The problem is both the absence of standard metrics and a generally accepted organization that could conduct such assessments. There is no Consumer Reports for Trustworthiness.**"

[Source: "Trust in Cyberspace," National Academy of Sciences report, National Academy Press, 1998.]

Regardless, There are Many!

- Hundreds of software engineering and software quality standards are in existence, but relatable? Good luck.
- Terminology: Methodologies, Taxonomies, Ontologies,
- Software standards are generally tied to: (1) risk management, (2) project management, (3) systems engineering, (4) software engineering, (5) languages, and (6) life-cycle phases and artifacts
- References:
  - Software Safety and Reliability, Debra S. Herrmann, IEEE Computer Society Press, 1999.
  - Software Engineering Standards, James W. Moore, IEEE Computer Society Press, 1998.
  - Guide to Software Engineering Standards and Specifications, Stan Magee and Leonard L. Tripp, Artech House, 1997.

Question 1:

Are Prescriptive Standards worth the costs?,i.e., Do They Offer Return on Investment?

Never discount the lowest common denominator problem.

Pros

Any bar or hurdle is better than no bar or hurdle

Cons

Possibly the developers would have done more to improve quality but now feel they have a license to do less.

Many organizations use them simply for “CYA” purposes

Question 2:

What Data Do You Have to Support Certifying Against a Standard

Information to support the creation of certificates should be based on a *claims-evidence-arguments* framework (Adelard, U.K.), much as is done in courts of law.

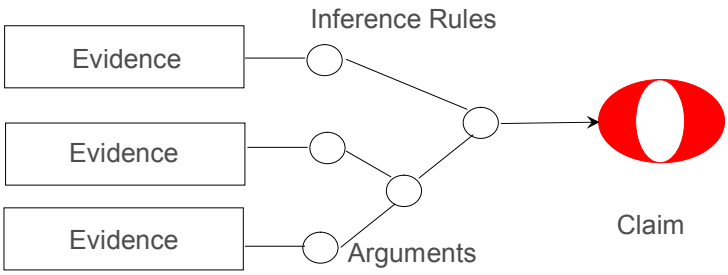
Goal Structuring Notation (U. of York, U.K.)

Arguments and Evidence and Claims

- *Supporting Evidence*: Results from observing, analyzing, testing, simulating and estimating the properties of a system that provide the fundamental information from which a *claim* (i.e., certificate) can be made
- *High Level Argument*: Explanation of how the available evidence can be reasonably interpreted as indicating acceptability for use (Fitness for Purpose), usually be demonstrating Compliance with Requirements
- Argument without Evidence is **unfounded**
- Evidence without Argument is **unexplained**

Basic Argument Structure

Claim: what we want to show
Argument: why we believe the claim is met, based on
Evidence: test results, other analysis results



Goal Structuring Notation

- Purpose of a goal structure: To show how goals are broken down into sub-goals, and eventually supported by evidence, while making clear the strategies adopted, the rationale for the approach (assumptions, justifications), and the context in which the goals are stated.
- Similar to a process flow chart
- Excellent for defining all processes that must be performed during development prior to contract award; thus useful for certification as well as acquisition.

Standards Are Not Perfect

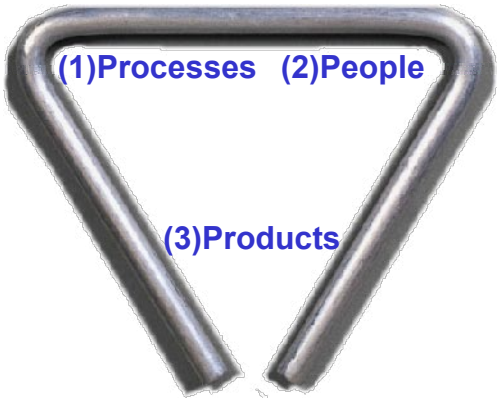
- Vague: Develop software that only does "good" things
  - Common sense "dos" and "don'ts" - Very watered done by voting time
- Disclaimers by publishing organizations
  - Profitable to organization that publishes them
- Used only if mandated
- Return-on-investment is often un-quantified
- Thwart intellectual creativity
  - "Protectionist" legislation
- Paperwork
  - 2167A: ~400 English words per Ada code statement
- "Old news" before being ratified (5-10 actual years)
- Relating one to another is very hard
  - Hundreds in existence

Standards are Not Perfect (cont)

- Different interpretations
- Process certifications are just documentation checks unless personnel remain on site during the assessment
- Re-certification
  - Client: over 300 modifications to a safety-critical medical device that never requested re-certification for any of those mods.
- Cannot be easily tested for compliance
  - Mis-certifications are common
- Lack of fairness during certification judgment
  - FDA Center for Devices and Radiological Health (CDRH)
- So much legacy functionality exists that complies with no standards yet still gets integrated, making it's impact to the system unknown.
  - WAAS

Three Schools of Thought

All certification approaches incorporate one or more of these perspectives



1. Process: Clean Pipes, Dirty Water?

2. People

A Fool with a Tool is Still a Fool!

3. Product: The Software Itself

0% confidence **100% confidence**

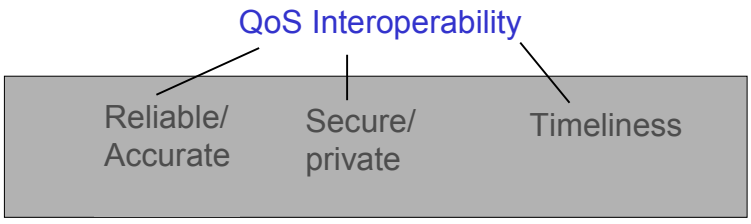
Spectrum of possibilities as to what a certificate proclaiming that some "quantified" level of quality has been built in could state --- it could say anything in the range between "Nothing" (*e.g.*, "here is a piece of software", etc.) to "This software will always work perfectly under all conditions" (i.e., a 100% guarantee of perfection).

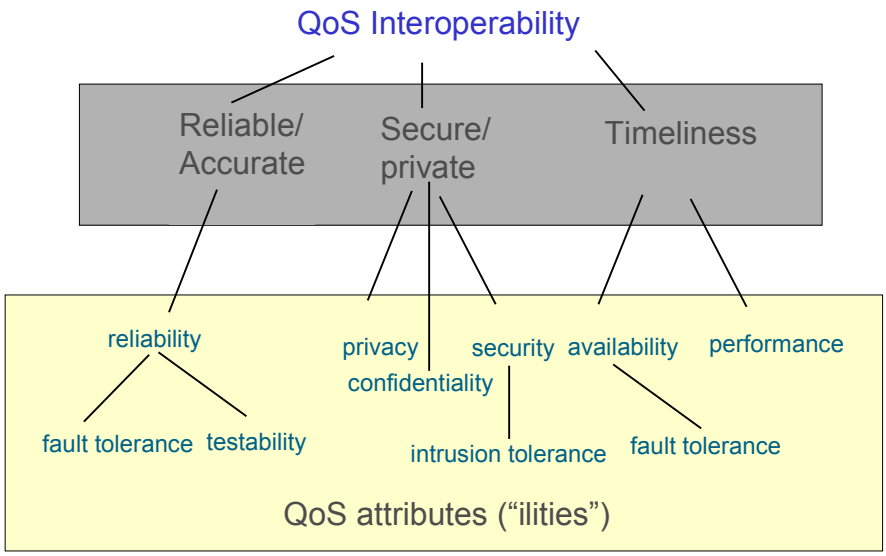
And So How Should a Standard Be Selected?

Four Principles

1. **People**, **Process**, **Product** view? With respect to an **Environment**?
2. Which **Attribute(s)** is of interest?
3. **When** in the Life-Cycle will the standard be applied, and to which artifacts?
4. **System/Enterprise Policies** that need to be enforced.

QoS Attributes





Position Statement

Software’s QoS interoperability is some combination of:

(1) the degree to which the *functional* requirements are met, as well as, (2) the degree to which the *non-functional* requirements are met.

Attributes Need to Be Pre-Defined

- Requirements should prescribe at some level of granularity as to what the weights are for various "ilities", as well as how much of each "ility" is desired.
- But HOW?
- Ignoring the attributes is not an option for achieving high assurance and trustworthy systems!

Weighting Is Important

w_1R w_2P w_3F w_4Sa
w_5Se w_6A w_7T w_8M

in order to not over-design any attribute into the system.

For example, for an web-based transaction processing application, w_4 would probably equal 0.0 and w_7 would also be less than something like w_2

Tradeoffs

How much will you spend for increased reliability knowing that doing so will take needed, financial resources away from security or performance or …?

- Security *vs.* Performance
- Fault tolerance *vs.* Testability
- Fault tolerance *vs.* Performance
- etc.

Counterintuitive Realities

- 100% safety and 0% reliability
- 100% reliability and 0% safety
- 0% functionality/reliability and 100% security
- 100% availability and 0% reliability
- 100% availability and 0% performance
- 0% performance and 100% safety

Common Sense Rules

- **Consider what you want demonstrated before selecting an existing standard**
  - **Know which "ilities" or assurances you are most concerned about**
- **Then, perform a risk/consequence analysis up front defining what failures modes are intolerable**
- **Consider the ROI of various standards**
- **There are hundreds of standards, and more are published/revised each year**
- **Do not assume good people = good code**
- **Do not assume good process = good code**
- **Work with qualified certification authorities or acquisition agents**
- **Recognize that documentation can be faked to fool a certifier**
- **No standard is perfect – blending is good**

Conclusions

- The 7 key components of the assurance problem well-bound the space.
- Software engineering standards are necessary but insufficient.
- The "ilities" are a huge piece of the puzzle

Contact Information

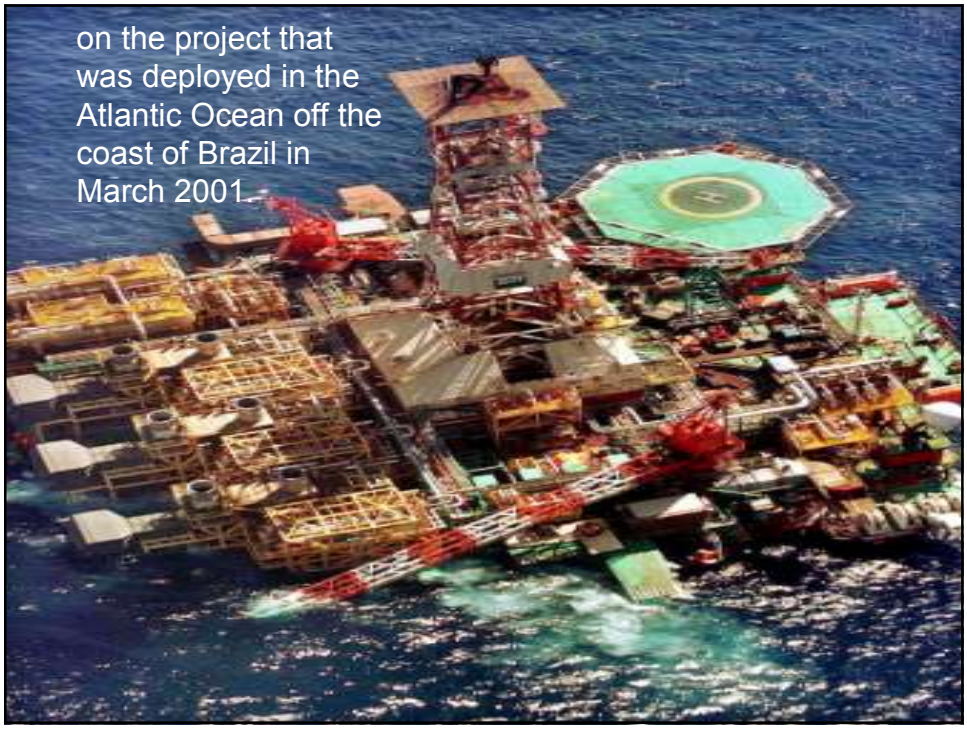
Jeffrey Voas, PhD
Director, Systems Assurance Technologies
SAIC
Crystal Gateway #4, 200 12th Street South, Suite 1500
Arlington, VA 22202
Phone: 703-414-3842
Fax: 703-414-8250
Email: jeffrey.m.voas@saic.com

QUALITY ASSURANCE AND THE SINKING OF THE LARGEST OFFSHORE OIL PLATFORM

March 2001


please read this quote from a
Petrobras executive,


extolling the benefits of
cutting quality assurance
and inspection costs,


on the project that
was deployed in the
Atlantic Ocean off the
coast of Brazil in
March 2001.


"Petrobras has established new global benchmarks
for the generation of exceptional shareholder wealth


through an aggressive and innovative program of
cost cutting on its P36 production facility.


Conventional constraints have been successfully challenged


and replaced with new paradigms appropriate to the globalized corporate market place.


Through an integrated network of facilitated workshops,

the project successfully rejected: (1) the established constricting and negative influences of prescriptive engineering,

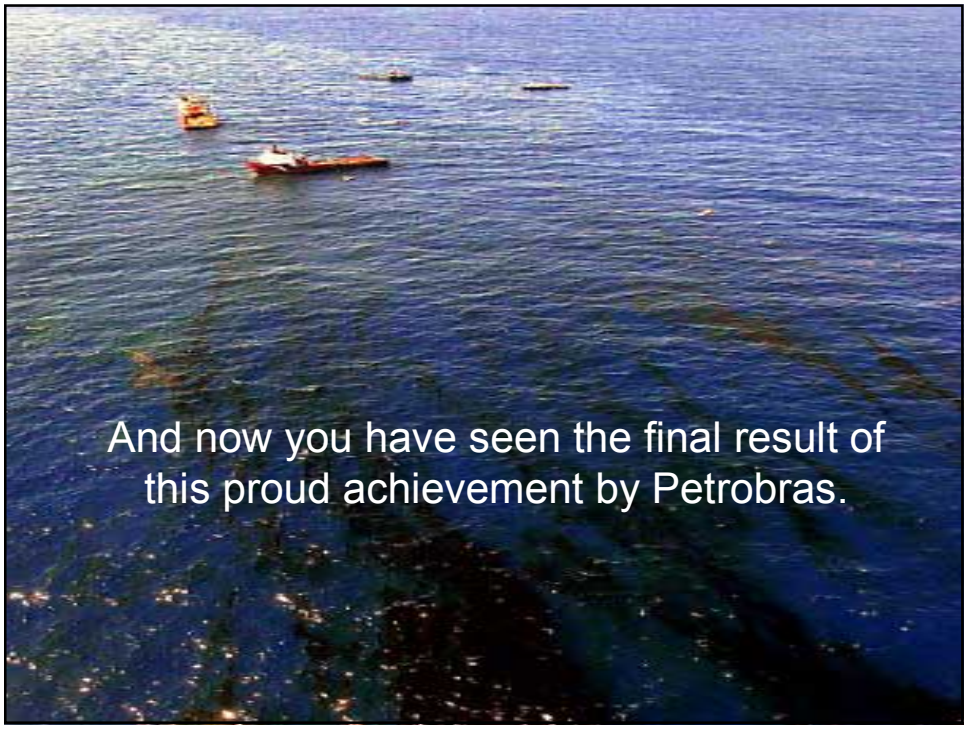
(2) onerous quality requirements, and (3) outdated concepts of inspection and client control.

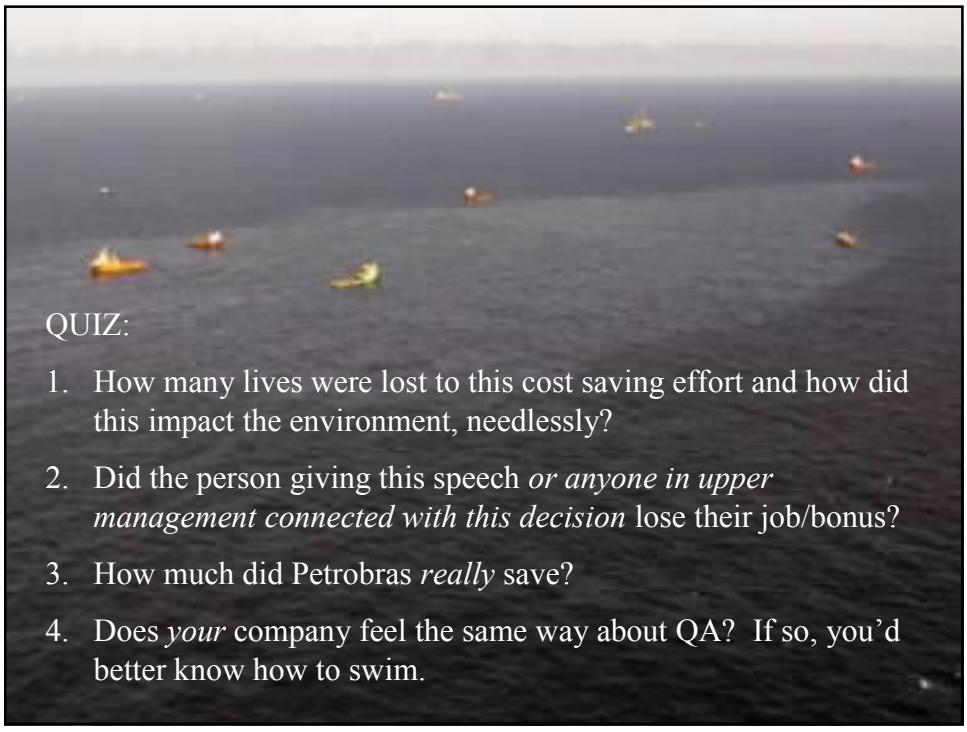

Elimination of these unnecessary straitjackets has empowered the project's suppliers and contractors to propose highly economical solutions,


with the win-win bonus of enhanced profitability margins for themselves.


The P36 platform shows the shape of things to come


in the unregulated global market economy of the 21st Century.”


And now you have seen the final result of
this proud achievement by Petrobras.


QUIZ:
1. How many lives were lost to this cost saving effort and how did this impact the environment, needlessly?
2. Did the person giving this speech *or anyone in upper management connected with this decision* lose their job/bonus?
3. How much did Petrobras *really* save?
4. Does *your* company feel the same way about QA? If so, you'd better know how to swim.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Major General Alan B. Salisbury, USA (Ret)
   Center for National Software Studies

4. Mr. John Harauz
   Jonic Systems Engineering, Inc.
   Willowdale, Ont., Canada

5. Dr. Sam Redwine
   James Madison University
   Harrisonburg, VA

6. Dr. Janusz Zalewski
   Florida Gulf Coast University
   Ft. Meyers, FL

7. Mr. Arch McKinlay
   Naval Ordnance Safety and Security Activity
   Indian Head, MD

8. Dr. Jeffrey Voas
   SAIC Inc.
   Arlington, VA

9. Mr. Brian Scannell
   Naval Surface Warfare Center
   Louisville, KY

10. Mr. Paul Dailey
    Naval Surface Warfare Center
    Louisville, KY

11. Dr. Gary Stoneburner
    Johns Hopkins University Applied Physics Laboratory
    Laurel, MD

12. Dr. Sylvain Dahan
    Japan

13. Dr. Man-Tak Shing
    Naval Postgraduate School
    Monterey, CA

14. Dr. Mikhail Auguston
    Naval Postgraduate School
    Monterey, CA

15. Major Bradley Warren
    Naval Postgraduate School
    Monterey, CA

16. Flight Lieutenant Patrick Redmond
    Royal Australian Air Force

17. LTC Thomas Cook
    Naval Postgraduate School
    Monterey, CA

18. Mr. Michael Brown
    EG&G Technical Services
    Fresno, CA

19. Dr. Butch Caffall
    NASA IV&V Facility
    Fairmont, WV

20. Dr. Michael Hinchey
    NASA Goddard Space Flight Center
    Greenbelt, MD

21. Dr. Bret Michael
    Naval Postgraduate School
    Monterey, CA